Department of Computer Science and Engineering

Towards Agile Product and Portfolio Management

Jarno Vähäniitty

Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 15/2012

Towards Agile Product and Portfolio Management

Jarno Vähäniitty

Doctoral dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the School of Science for public examination and debate in Auditorium TU1 at the Aalto University School of Science (Espoo, Finland) on the 10th of February 2012 at noon.

Aalto University
School of Science
Department of Computer Science and Engineering
Software Process Research Group

Supervisor
Professor Casper Lassenius

Instructor
Professor Casper Lassenius

Preliminary examiners
Professor Björn Regnell
Lund University, Sweden

Professor Pasi Tyrväinen
University of Jyväskylä, Finland

Opponents
Professor Björn Regnell
Lund University, Sweden

Postdoctoral researcher Petri Kettunen
Helsinki University, Finland

Aalto University publication series
DOCTORAL DISSERTATIONS 15/2012

ISBN 978-952-60-4505-4 (printed)
ISBN 978-952-60-4506-1 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Unigrafia Oy
Helsinki 2012

Finland

The dissertation can be read at http://lib.tkk.fi/Diss/

NORDIC ECOLABEL
441 697
Printed matter

Author
Jarno Vähäniitty

Name of the doctoral dissertation
Towards Agile Product and Portfolio Management

Publisher School of Science

Unit Department of Computer Science and Engineering

Series Aalto University publication series DOCTORAL DISSERTATIONS 15/2012

Field of research Software Engineering

Manuscript submitted 14 June 2011 **Manuscript revised** 11 November 2011

Date of the defence 10 February 2012 **Language** English

☐ **Monograph** ☒ **Article dissertation (summary + original articles)**

Abstract

Small growing software enterprises are an increasingly important source of innovation and employment. They strive to productize the technologies that enable their key business idea(s), but often offer professional services and custom development projects as well to balance cash flow and share risk. This requires the integration of long-term product and business planning with modern, flexible but controlled approaches to software development promoted by the agile/lean software development movement.

There is still little empirical research on agile software development, and most of the practitioner literature has concentrated on the perspective of a single team in an individual development project, leaving the links to product and portfolio management largely unaddressed. Likewise, existing literature on product and portfolio management essentially views development as an activity that can be planned in advance and then executed according to the plan. This gap in theory is problematic for small software organizations who, in order to remain operationally effective, need to maintain the big picture of the ongoing work of the development staff and align this with the long-term plans of the enterprise.

This dissertation summarizes existing and presents new understanding for linking product and portfolio management with modern development methodologies such as Scrum in the face of the practical realities that may apply to many small software organizations. The research approach taken is that of design science and constructive research. The results are based on both findings from qualitative, participative action research -type case studies and a synthesis of related work based on a systematic review of research and practitioner literature.

As results, we propose a framework that shows how the three key processes that should connect business and development decision-making – product roadmapping, release planning and different levels of portfolio management – can be understood in the context of organizations striving for agile software development. We also present an example of how a product roadmap can be visualized, state that explicit portfolio management is under certain conditions crucial for small organizations as well, and provide guidelines for it.

As proof-of-concept, we present Agilefant (www.agilefant.org), an open source support tool for managing a portfolio of activities of which some – though not necessarily all – are planned and managed using backlogs with hierarchical work item structures. We propose that this provides transparency to business priorities while still enabling just-in-time elaboration required by agile software development.

Keywords agile, product backlog, software development, product management, portfolio management, Scrum

ISBN (printed) 978-952-60-4505-4 **ISBN (pdf)** 978-952-60-4506-1

ISSN-L 1799-4934 **ISSN (printed)** 1799-4934 **ISSN (pdf)** 1799-4942

Location of publisher Espoo **Location of printing** Helsinki **Year** 2012

Pages 215 **The dissertation can be read at** http://lib.tkk.fi/Diss/

Aalto-yliopisto

Tiivistelmä

Aalto-yliopisto, PL 11000, 00076 Aalto www.aalto.fi

Tekijä
Jarno Vähäniitty

Väitöskirjan nimi
Kohti ketterää ohjelmistotuotteen ja -tekemissalkun hallintaa

Julkaisija Perustieteiden korkeakoulu

Yksikkö Tietotekniikan laitos

Sarja Aalto University publication series DOCTORAL DISSERTATIONS 15/2012

Tutkimusala Ohjelmistotuotanto

Käsikirjoituksen pvm 14.06.2011 **Korjatun käsikirjoituksen pvm** 11.11.2011

Väitöspäivä 10.02.2012 **Kieli** Englanti

☐ **Monografia** ☒ **Yhdistelmäväitöskirja (yhteenveto-osa + erillisartikkelit)**

Tiivistelmä

Pienet, kasvavat ohjelmistoyritykset ovat entistä tärkeämpiä työllisyydelle ja teollisuuden innovaatiokyvylle. Pienten ohjelmistoyritysten erityishaasteena on tasapainoilu tuote- ja palveluliiketoiminnan välillä. Siinä onnistuminen vaatii kuitenkin kasvavissa määrin kykyä yhdistää pitkän tähtäimen tuote- ja liiketoimintasuunnittelua moderneihin, ns. ketterän ohjelmistokehityksen menetelmiin.

Ketterästä ohjelmistokehityksestä on toistaiseksi vain vähän korkealaatuista tutkimustietoa. Alan kirjallisuus on tähän mennessä käsitellyt ketteriä menetelmiä enimmäkseen yksittäisen kehitystiimin ja/tai ohjelmistoprojektin näkökulmasta. Käytännön haasteeksi nousee kuitenkin ketterien menetelmien yhdistäminen tuotehallintoon sekä moniprojektiympäristön johtamiseen – joita käsittelevä kirjallisuus jättää ketterät menetelmät pitkälti huomiotta. Käynnissä olevien projektien ja muun toiminnan kokonaiskuvan hahmottaminen ja liiketoiminnallisesti tarkoituksenmukainen priorisointi ovat kuitenkin keskeisiä käytännön haasteita monissa pienissä ohjelmisto-organisaatioissa.

Tässä väitöskirjassa käsitellään tuotehallinnon ja projektisalkun hallinnan roolia ketterissä ohjelmistokehitysmenetelmissä (esim. Scrum) pienten ohjelmisto-organisaatioiden erityispiirteitä huomioiden. Aiheen käytännönläheisyyden vuoksi tutkimuksen empiiriseksi lähestymistavaksi valittiin konstruktiivinen, laadullinen osallistuva toimintatutkimus joukossa tapausyrityksiä. Tulosten positioinniksi suoritimme systemaattisen kirjallisuuskatsauksen joka tieteellisen kirjallisuuden lisäksi kattoi myös ammatinharjoittajille suunnattua kirjallisuutta.

Tutkimuksen tuloksena syntyi malli joka esittää miten liiketoiminta- ja ohjelmistokehitystä yhdistävät avainprosessit – tuotekehityksen tiekartoitus, tuotejulkaisujen suunnittelu sekä tuote- ja projektisalkun hallinnan eri tasot – voidaan ymmärtää kohti ketterää ohjelmistokehitystä pyrkivissä organisaatioissa. Esitämme myös esimerkin tuotekehityksen tiekartasta, selitämme miksi tietoinen ”tekemissalkun” hallinta on ainakin tietyissä tilanteissa keskeistä myös pienille organisaatioille ja tarjoamme ohjeita siihen.

Mallin toiminnalliseksi validoinniksi työssä esitellään Agilefant (www.agilefant.org), avoimen lähdekoodin työkalu ketteriä menetelmiä kokonaan tai osittain noudattavan organisaation töiden hallintaan. Agilefant täydentää ketterien menetelmien mukaista työlista-ajattelua mahdollistamalla myös työn osituksen hierarkkisia rakenteita käyttäen. Esitämme, että tämä edistää liiketoiminnallisten tavoitteiden näkyvyyttä sallien myös ketterien menetelmien mukaisen vaatimusten juuri-oikeaan-tarpeeseen tarkentamisen.

Avainsanat ketterät menetelmät, tuotteen kehitysjono, ohjelmistokehitys, projektisalkun hallinta, tuotteenhallinta, Scrum

ISBN (painettu) 978-952-60-4505-4 **ISBN (pdf)** 978-952-60-4506-1

ISSN-L 1799-4934 **ISSN (painettu)** 1799-4934 **ISSN (pdf)** 1799-4942

Julkaisupaikka Espoo **Painopaikka** Helsinki **Vuosi** 2012

Sivumäärä 215 **Luettavissa verkossa osoitteessa** http://lib.tkk.fi/Diss/

ACKNOWLEDGEMENTS

This research was done in the Software Process Research Group at SoberIT, Department of Computer Science and Engineering at Aalto University. I would like to extend my thanks to those who have funded this research: The Finnish Funding Agency for Technology and Innovation (Tekes), and the companies who participated in the Lucos, SEMS, SHAPE and ATMAN research projects and The Cloud Software Program, the Graduate School for Business and Software industry (GEBSI) and Graduate School on Software Systems and Engineering (SoSE).

I would like to thank Professors Casper Lassenius (my supervisor) and Reijo "Shosta" Sulonen for guiding my work and providing timely and thoughtful feedback throughout the years. As the preliminary examiners, the comments from Professors Björn Regnell and Pasi Tyrväinen were invaluable in the final tuning of the manuscript.

Ville Heikkilä, Juha Itkonen, Timo Lehtinen, Mika Mäntylä, Tuomas Niinimäki, Maria Paasivaara, Arttu Piri, Kristian Rautiainen and Jari Vanhanen of the SPRG research group deserve many thanks. They have made it possible for me to concentrate on writing, and have been of crucial importance because of their daily support and comments – not to mention upholding the fun, friendly and easy-going working environment we have.

I am deeply grateful to the people who have during the years contributed to Agilefant's development. While they are too many to list here, Ilkka Lehto, Reko Jokelainen, Pasi Pekkanen, Antti Haapala, and Aleksi Toivonen deserve special mention, as do the student teams 2rox, Maranello, Spider, Testarossa and SuperAmerica (including Professor Daniela Damian at Victoria University, Canada). The ongoing pursuit of the Simplest Solution that Scales owes everything to them.

I also wish to thank Professor Sjaak Brinkkemper of Utrecht University and his research group for providing me a haven for getting the writing of the summary part started, and then doing this again for the practitioner book (that not-so-coincidentally has the same title as this Ph.D.) we put together in the last two months of the ATMAN research project.

Last, but definitely not least, I wish to thank my lovely wife Paula and our wonderful daughters Ella and Isla who are invaluable (among other things) in keeping my own portfolio balanced. I also want to thank my parents for their support and encouragement.

Espoo, January 2012,

Jarno Vähäniitty

jarno@agilefant.org

LIST OF PUBLICATIONS

Below is a list of the publications included in this dissertation. The contributions are explained in more detail below.

I. Vähäniitty, Jarno. "A Tentative Framework for Connecting Long-Term Business and Product Planning with Iterative & Incremental Software Product Development". In proceedings of the 7th International Workshop on Economic-Driven Software Engineering Research (EDSER-7) at ICSE 2005, St. Louis, USA, 2005

II. Vähäniitty, Jarno. "Key Decisions in Strategic New Product Development for Small Software Product Businesses". In proceedings of Euromicro 2003. Antalya, Turkey, 2003

III. Vähäniitty, Lassenius, Rautiainen & Pekkanen: "Long-term Planning of Development Efforts by Roadmapping - a Model and Experiences from Small Software Companies". In proceedings of Euromicro 2009. Sep 2009, Patras, Greece

IV. Vähäniitty, Rautiainen & Lassenius: "Small Software Organizations Need Explicit Portfolio Management" IBM journal of Research and Development", Vol. 54, No. 2 (Issue on Creating Business Value through Software Development), April 2010

V. Vähäniitty, J & Rautiainen, K. "Towards an Approach for Development Portfolio Management in Small Product-Oriented Software Companies". In proceedings of Hawaii International Conference on System Sciences (HICSS-38), Jan 2005, Big Island, Hawaii, USA.

VI. Vähäniitty & Rautiainen: "Towards a Conceptual Framework and Tool Support for Linking Long-term Product and Business Planning with Agile Software Development". ICSE International Workshop on Software Development Governance (SDG2008), May 2008, Leipzig, Germany.

The publications' relationships to each other, the research problem and the research questions are depicted in Figure 3.1 on page 59.

For all of the publications, the author of this dissertation is the primary author and responsible for all the research reported and all the text, excluding some editing. The other authors have helped shape the argumentation of the publications and given instructions and improvement suggestions in the research described in the publications.

For publication III, Suvi Elonen, Lassi Lindblom, Ilkka Miettinen and Pasi Pekkanen assisted the author in the literature review. Suvi Elonen, Lassi Lindblom and Ilkka Miettinen have also participated in creating an initial version of the roadmap visualisation presented in publication III (see section 3.3.2 for details).

Agilefant, discussed in publication VI and section 4.5, is the result of joint design and development efforts undertaken in the ATMAN research project and the Cloud Software Program coordinated by Jarno Vähäniitty as well as other researchers from the Software Process Research Group at SoberIT, Aalto University; for a full list of contributors, see www.agilefant.org.

Jarno Vähäniitty is the primary author and responsible for all of the text in this summary. Parts of this summary have previously been published in the book "Towards Agile Product and Portfolio Management" that summarizes the results of the ATMAN research project and is aimed for a practical audience.

Table of Contents

Index of Figures

Index of Tables

1 Introduction

In this chapter, the background, the motivation and the objectives of this dissertation are described. The chapter concludes with an outline of the structure of the dissertation.

1.1 Background

In the industrialized world, small – less than 50 employees (Richardson & von Vangenheim 2007) – growing high tech companies are an increasingly important source of employment and significantly contribute to the gross domestic product (Chaston 2009). Such companies are important for international competitiveness, innovativeness and regional economic regeneration especially during the early stages of new, emerging technologies (Richardson & von Vangenheim 2007, Berry 1996, Miettinen, Mazhelis & Luoma 2010). Small software companies provide products and services that enable growth of other industries (Miettinen, Mazhelis & Luoma 2010), and the software industry in most countries has an "industrial backcloth" that is made up mainly of small and medium software enterprises (Pino, Garcia & Piattini 2008).

The majority of small software enterprises either do not wish, or simply are not able to acquire significant external funding (Rönkkö et al. 2009). Out of the internally funded companies, those who wish to grow profitably are 'hybrids' out of necessity (Miettinen, Mazhelis & Luoma 2010, Cusumano 2003, Wangenheim et al. 2006). This means that besides striving to productize the technologies their key business idea(s) rest upon, they on the side have to offer professional services and custom development projects – which may not contribute to the productization efforts at all – to balance their cash flow and share risk (Cusumano 2004, Artz et al. 2010, Cusumano 2008).

Moreover, the processes, competencies and resources needed for effectively running product-based and project/service-based software businesses are intrinsically different (Artz et al. 2010, Nambisan 2001). The set of technical and organizational capabilities required to run a hybrid company has been referred to as "daunting" and "hard to master", with an improper balance between product development and servicing efforts being "an easy way to ruin an otherwise good business" (Cusumano 2004, Alajoutsijärvi, Mannermaa & Tikkanen 2000).

Thus, even with respect to their internal processes, small, internally funded hybrid software companies that are intent on growing have to excel on three fronts: product management, multi-project management and software development. This is further explained in the following paragraphs as well as illustrated in Figure 1.1 on page 2.

First, the business environment seems to be getting increasingly challenging for small software companies to prosper. While the software industry is growing relatively rapidly (Wangenheim et al. 2006), it is undergoing structural changes that challenge the traditional ways business has been conducted (Koivisto 2010). Product-based software business, once regarded as the chance for the Finnish software industry to gain a significant position in global markets, may no longer provide opportunities for growth (Rönkkö et al. 2009). Instead, success and growth today are perceived to be driven by business model innovation and the ability to successfully navigate the turbulent business environment (Ktata & Levesque 2009, Vidgen & Wang 2009). Moreover, the ongoing convergence of product and service business models presents both opportunities as well as threats (Rönkkö et al. 2009, Cusumano 2008). These factors make overcoming the classic challenge of integrating long-term business, product and release planning with technology development (Berry & Taggart 1998) increasingly crucial. While the future and the available business opportunities seem to change so rapidly that long-term planning may at times feel pointless (Doz & Kosonen 2008), the situation calls for excellence in product management.

Second, while long-term product and business goals should set the framework for taking action, short-term cash flow and customer satisfaction cannot be neglected. In order to make progress in productization[1], small companies must be able to "juggle" their scarce resources so that at each moment those activities that from a business perspective are the most important get attended to (Miettinen, Mazhelis & Luoma 2010). Managing this kind of "juggling" is traditionally referred to *multi-project management* (Elonen & Artto 2003).

Third, development must possess the capability to quickly respond to emerging opportunities and market demand (Rönkkö et al. 2009, Ktata & Levesque 2009, Cusumano et al. 2009). To make this possible, the employed development processes must be flexible, controlled and driven by customer needs (Takeuchi and Nonaka 1986; MacCormack, Verganti & Iansiti 2001). Approaches to software development that are claimed to possess these qualities have in the recent years been proposed by the agile software development movement (Smith 2008).

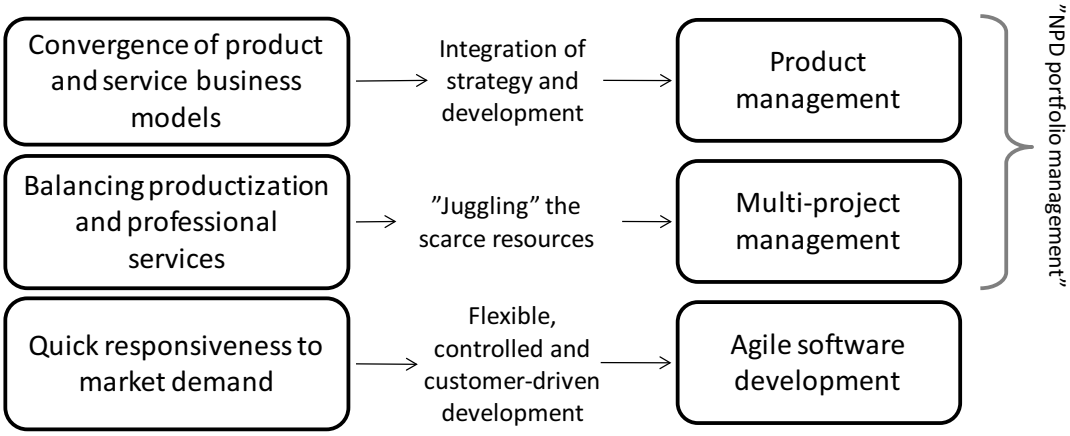


Figure 1.1 Demands for small growth companies' internal processes in today's software industry

Thus, in order to respond to the demands posed by today's software industry, small companies intent on growing would have to be complement agile development processes with top-class product and multi-project product management.

1.2 Motivation

This dissertation is motivated by two facts. First, the theoretical links between agile software development and product management are at best unclear. Second, existing literature on product and as well as NPD portfolio management apply to the small organization and agile software development contexts equally poorly. These challenges are illustrated in Figure 1.2 and further explained below. An in-depth discussion follows in chapter 2 (*Related work*).

[1] Here, *productization* can refer to increasing the degree of productization of a software offering, but also to a 'productization' of a service (Cusumano 2008). Also, servicing may be complementary to the current products the company is offering, but it can also be completely independent from the product portfolio, and may in turn create new needs for productization and innovation opportunities (Ruokonen 2008).

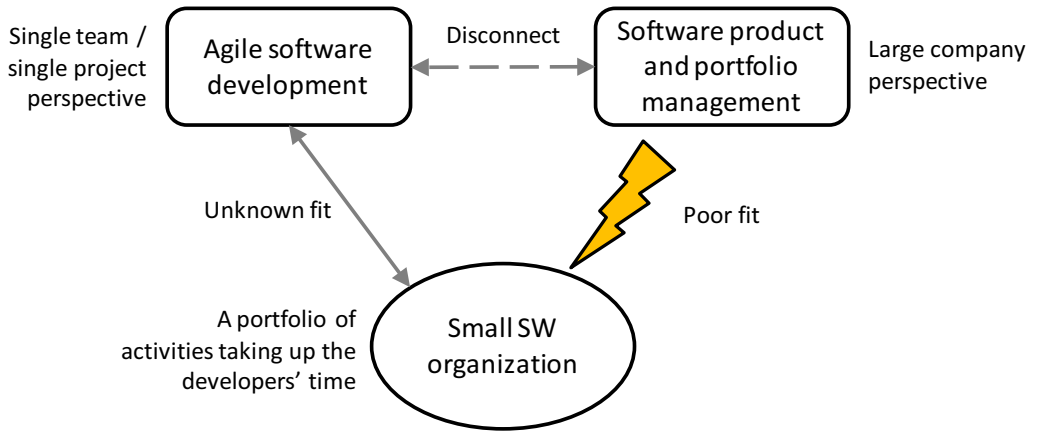


Figure 1.2 Product and portfolio management literature are disconnected from agile software development, and the single project perspective taken in agile has an unknown fit to the small organization context

Agile software development seems to be increasingly popular among practitioners (Dybå & Dingsøyr 2008). While some amazing results have been reported (Sutherland & Altman 2010) – even in challenging, distributed settings (Sutherland, Schoonheim & Mauritz 2009) and with outsourced development teams (Sutherland et al. 2007) – not many can to date present such impressive findings. Overall, empirical evidence regarding agile methods' effectiveness is scarce and largely anecdotal (Dybå & Dingsøyr 2008). However, the little evidence that exists can be considered encouraging (Cardozo et al. 2010, Syed-Abdullah, Holcombe & Gheorge 2006, Petersen & Wohlin 2010).

Small companies could reasonably be viewed a fertile soil for agile methods to take root (Wangenheim et al. 2006). However, literature on agile software development has, along with the majority of software engineering literature (Glass, Vessey & Ramesh 2002), focused on a single-team-single-customer-single-product setting (Shalloway, Beaver & Trott 2009). How agile software development should link with the running of an entire company has not been discussed (Kettunen & Laanti 2006, Kennaley 2010). Likewise, how agile software development should link with product and release planning (Valkenhoef et al. 2010) or portfolio management (Kettunen 2007) has not been discussed.

Overall, there seems to be little industry awareness of the impact that agile software development has or should have on long-term strategic product planning or portfolio management (Kalliney 2009). It has been specifically argued that agile methods should be extended to better address product and release planning (Valkenhoef et al. 2010) and "longer-term product evolution and portfolio management" (Kettunen & Laanti 2006). However, empirical research on this is still scarce (Lehto & Rautiainen 2009), although some experience reports exist. Furthermore, the approaches prescribed in the literature for long-term product and business planning or portfolio management seem to originate from a large company context (Jennings & Beaver 1996, Martinsuo 2001) and do little to utilise the strengths inherent to small companies such as flexibility, quick responsiveness and informal but direct communication structures (Wangenheim et al. 2006, Beattie & Fleck 2005, Pino et al. 2010a).

1.3 Objectives

For an organization as a whole to be agile, as opposed to just software development, business

practices have to evolve as well (Vidgen & Wang 2009). To support this and to aid small organizations that develop software grow successfully, this dissertation presents **a framework** and a **proof-of-concept support tool** (Agilefant, www.agilefant.org) to help clarify how agile software development could be linked with product and portfolio management. As will be explained in chapter 4 (*Results*), the framework links three core process areas of software product management – portfolio management, product roadmapping, and release planning (Weerd et al. 2006b, Bekkers et al. 2010) – with agile software development as exemplified by Scrum (Schwaber & Beedle 2002).

In addition, this dissertation provides practical guidelines and summarizes lessons learned regarding roadmapping, release planning, portfolio management and agile software development in the context of small, relatively independent organizations that develop software and provide related services and strive for agile software development in at least some of their activities. Many of the lessons learned originate from situations in which only a part of the ongoing activities that require attention from the development people are following an agile life cycle – or even are conducted as explicit projects. As will be explained, such scenarios, though rarely addressed in the literature, may be fairly common in practice.

1.4 Scope

Because of the characteristics of our case organizations (see section 3.2 *Research approach and case companies*), this study is limited to discussing the context of small software organizations intent on growing, applying agile software methods in at least some (but not necessarily all) of their activities and which are open for process improvement collaboration with outside experts.

For more details, see the limitations of this study discussed throughout the *Results and discussion* and *Conclusion* chapters for a detailed discussion.

1.5 Outline

Chapter 2 (*Related work*) discusses existing work related to the topic and provides more thorough definitions of the key concepts of the study. Chapter 3 (*Research problem and methods*) describes the research problem, research questions and the research methods used. Chapter 4 (*Results and discussion*) summarizes the results of this thesis, discusses limitations and answers the research questions. Chapter 5 (*Conclusion*) concludes the summary part of this dissertation. In chapter 5 we answer the research problem in the light of the limitations, summarize the theoretical contribution, discuss its implications for practice, and provide directions for future research.

In addition to this summary part, the dissertation consists of six research papers published between 2003 and 2010. These are listed on page iii.

2 Related work

This chapter discusses the relevant existing literature related to this dissertation. First, the research space and the review method are briefly discussed (section 2.1). The sections that follow examine existing work regarding product (section 2.2) and portfolio (2.3) management and their relationship with agile software development. Then, implications of adopting agile methods in small, growth-oriented software companies are briefly discussed in the light of existing literature. The chapter concludes with a summary of the related work (section 2.5).

2.1 Research space and review method

This review of the literature has been conducted based on the assumption that existing work related to linking product and portfolio management with agile software development can to sufficient degree be found from examining engineering and management literature. The broad areas of literature that rise for closer examination are *software engineering* (SE) and *management of new product development* (NPD).

The Merriam-Webster online dictionary defines **engineering** as "the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people"[2]. **Software engineering** is an area of engineering literature that deals with processes, methods and tools that aim for enabling complex computer-based systems to be built in a timely manner with quality (Pressman 2010).

According to the Merriam-Webster online dictionary, **management** means "conducting or supervising of something as a business"[3]. **New product development** (NPD) is an area of management literature concerned with describing the overall process of strategy, organization, concept generation, product and marketing plan creation and evaluation and commercialization of new products (Kahn, Castellion & Griffin 2005). The concerns of software engineering literature partly overlap with NPD literature (Nambisan & Wilemon 2000).

In this research space, of particular interest are literature on **product management** and **portfolio management**, and in particular, the relationship of these processes with **lean/agile approaches to software development**. This is further discussed in sections 2.1.1-2.1.3 below.

In reviewing the related work, we sought for both scholarly and practitioner literature using a systematic approach inspired by the work of Petticrew & Roberts (2005), Hart (1998) and Kitchenham et al. (2009), and adapted to fit the resources and schedule available for completing this dissertation. For a longer explanation of the respective research question and the exact method used, see section 3.3.6 (*Approach and methods for* answering RQ6) in chapter 3 (*Research problem and methods*) on pages 73-78.

In discussing the related work, we have attempted to bring forth whether the cited material is of scholarly origin or not. However, only few – if any – of the discussed matters have been extensively researched and there hardly is a solid scientific basis for claims made either way[4], we consider this level of separating research publications from trade press literature and experience reports to be

[2] http://www.merriam-webster.com/dictionary/engineering

[3] http://www.merriam-webster.com/dictionary/management

[4] For example, whether a possible idea pool should be kept separate from the product backlog or not (see p. 23)

adequate.

2.1.1 Software product management

Product management deals with the planning, development and marketing of a product or products at all stages of the product lifecycle, spanning both strategic to tactical activities (Kahn, Castellion & Griffin 2005). While product management as a topic originates from management literature, there is an emerging body of literature in the field of software engineering that specifically deals with **software product management**. Software product management has been defined as *the process that governs a product/service offering from its inception to the market or customer delivery and service* (Ebert 2009). The core of the engineering aspects of software product management has been defined to consist of defining products, releases and managing requirements in collaboration with many internal (such as sales & marketing and development) and external (such as the customers and partner companies) stakeholders (Ebert 2009, Kittlaus & Clough 2009).

For the purposes of this dissertation we assume that the body of software product management literature covers the same issues as 'generic' product management literature, but is more specific to the field of this dissertation and thus more applicable. Thus, the discussion is restricted to existing work regarding software product management, and other product management literature is (mostly) out of scope. Existing work related to the topic of this dissertation that falls under the area of software product management, lean/agile or otherwise, is examined in section 2.2 (pages 9-39).

2.1.2 Portfolio management

As we will see in section 2.2 (*Product management*), in the software product management literature the term 'portfolio management' is used to refer to *product portfolio management*. Thus, it does not address perspective of managing multiple simultaneous activities such as ongoing development projects or other efforts that take up the development personnel's time. From the perspective of this dissertation, this aspect of project portfolio management is at least as important for a running even a small development organization (see publications IV and V) as product management.

In the literature on new product development, **portfolio management of new product development projects** – or (project) portfolio management for short – refers to the *process for achieving balanced resource allocation in terms of value maximization, strategic alignment, risk level and the number of ongoing projects* (Cooper, Edgett & Kleinschmidt 2002). A term that is closely related to portfolio management is **pipeline management**. Pipeline management "integrates product strategy, project management and functional management to continually optimize the cross-project management of all development-related activities" (Kahn, Castellion & Griffin 2005).

Thus, while discussing product management in the NPD literature is out of scope for this dissertation, some material from NPD regarding portfolio and pipeline management may be relevant, and thus are included in the research space.

Note, that in this dissertation the author has in many places chosen to omit the term 'project', and simply refer to 'portfolio management'. As suggested by the above definitions of portfolio and pipeline management, the scope of 'project portfolio management' should be considered wider than just addressing those activities that are explicitly defined or conducted as projects. This will also become evident later on in this chapter, as well as in the included publications.

Existing work related to this dissertation that falls under the area of portfolio management, agile or otherwise, is examined in section 2.3 (pages 39-52).

2.1.3 Agile and lean software development

The **agile software development** movement is concerned with software development methodologies that are based on iterative development and where requirements and solutions evolve through collaboration between self-organizing cross-functional teams and the customer (Smith 2008). The term 'agile software development' was coined in the year 2001 when the Agile Manifesto was formulated (Cohen, Lindvall & Costa 2005). There is no universally accepted definition of what agile software development entails (Kettunen & Laanti 2006). However, most approaches that are considered agile possess the following characteristics (Smith 2008, Aguanno 2005):

- The development proceeds iteratively and incrementally in loops of one to six weeks
- Progress is measured via completed features; each iteration should deliver working software
- New functionality is not considered 'done' until it has been integrated as part of the whole
- Product requirements are re-assessed and re-prioritized at the end of each iteration
- Personal communication; for example the (representative of the) customer is incorporated in planning
- Small, close-knit, cross-functional empowered and preferably co-located teams do the work
- Striving for open, flexible design

In general, agile software development emphasizes building releasable software in short, fixed time periods and emphasises flexibility, communication, collaboration and working software over processes, tools, documentation and following a pre-defined plan (Rico, Sayani & Sone 2009). The fundamental belief that project outcomes cannot be predicted is said to be that which most sets agile software development apart from traditional project management philosophies (Aguanno 2005).

The two dominant agile software development methods, Scrum and XP, were born somewhat independently from "developer-centric frustrations with tool vendors, ivory tower methodologists, […] command & control management and big up-front design typical of the Waterfall approach" (Kennaley 2010).

Before agile methodologies, similar challenges have been earlier addressed in **agile manufacturing** (Kettunen 2009) and **lean production systems** (Coplien & Bjørnvig 2010). In fact, Lean is often cited as a foundation of Agile, but upon a closer look, both Lean and Agile can be seen to arise from complex adaptive systems theory (Coplien & Bjørnvig 2010). For example, according to Sutherland (2008), the Scrum framework was directly inspired by the "New New Product Development Game" article by Takeuchi and Nonaka (1986).

Although agile software development and lean production have profound differences, many parallels also exist, and the two approaches can be seen not only contrasting but complementary as well (Kettunen 2009, Coplien & Bjørnvig 2010). While the similarities between manufacturing and software development may not be immediately obvious, they are said to become "startlingly clear" when *decisions* and *decision-making* in an organization developing software are reflected upon as of as the "inventory" of a production system (Cockburn 2007). Moreover, as much of the current agile software development methods directly address only software development, there is much potential for software organizations to improve by seeing agility and leanness as something the entire organization should strive for (Coplien & Bjørnvig 2010).

Thus, in this review of the literature we are interested in both Agile as well as Lean approaches for linking software development with product and/or portfolio management.

2.1.4 Definitions and glossary

As the sections that follow, especially 2.2 (*Product management*) and 2.3 (*Portfolio management*) review existing literature, they contain many terms and definitions. For some terms, we present multiple existing definitions, some of which are in conflict.

To keep the discussion of the related work concise and to refrain from using terms without a prior definition, we have collected definitions for some concepts commonly used in the context of agile software development in Table 1 below. For some terms, multiple definitions may be provided; in these cases, we have seen the definitions complementary and mutually compatible.

The definitions in Table 1 *apply at least when the concept is first mentioned*. Because of the conceptual and constructive nature of this study, some of the terms are re-defined over the course of chapter 2 as the discussion progress; these are marked with an asterisk (*). When alternate definitions are discussed, this will be obvious in the context. The "final definitions" of all of the key terms used in this dissertation have been collected in Appendix A (Glossary).

Table 1 Definitions of selected terms

Term	Definition(s)
Burndown / Burn-up	"Burndown (and/or) burn-up charts show how much work remains to be completed in an iteration or a release, tracked against time." (Cohen 2010), p. 125
Enterprise	An organization engaging in commercial activity to make profit by delivering value to its customers and the society
*Epic	"A large user story is sometimes called an epic." (Cohn 2005), p. 53; "[Epics are] bold, impactful, marketable differentiators" (Leffingwell 2011) p. 455
*Feature	"Short, descriptive, value delivery and benefit-oriented statement [of system functionality]" (Leffingwell 2011) p. 455
Organization	An organized group of people with a particular purpose (Oxford dictionaries online, http://oxforddictionaries.com/definition/organization, referenced 20.9.2011); used as a generic term for 'enterprise'
Product owner	"The person who is responsible for what the Scrum Team builds and for optimizing the value of it." (Schwaber 2007), p. 115
*Product backlog	"A prioritized list of functional and non-functional requirements and features to be developed for a new product or to be added to an existing product." (Schwaber 2007), p. 114
*Product vision	"The highest level of strategy that is easy to articulate and covers the target customer, their pain point, and how the solution will evolve to solve it." (Cohen 2010), p. 127
Project	A non-routine undertaking that has specific objectives, a pre-determined time span and involves more than one person (Hughes and Cotterell 2002)
Scrum	"A process for managing the development and deployment of complex products that is based on empirical process control theory and stands on the core practices of iterative development, which generates increments of product by using self-managing, cross-functional teams" (Schwaber 2007), p.115
Sprint	"A Scrum iteration, normally of a one-month duration, but shorter durations can be used" (Schwaber 2007), p. 115
User story (or story)	"Burndown (and/or) burn-up charts show how much work remains to be completed […], tracked against time." (Cohen 2010), p. 125

2.2 Product management

Linking business and development is a classical problem (Griffin & Hauser 1996), the tackling of which can essentially be considered the duty of product management. This entails, among other things, understanding the market, conducting long-term planning, and directing the development according to the strategic, long-term product and business goals.

In the following subsections, we discuss existing work on software product management, and in particular, those of its aspects that relate to "longer-term planning of product evolution" as these are, when it comes to agile software development, considered to be missing or at least implicit (Kettunen & Laanti 2006, Mohan, Ramesh & Sugumaran 2010). First we discuss current research on software product management as represented by the *software product management reference framework* (2.2.1). Then, we focus on two key areas of the reference framework from the perspective of linking long-term planning with fast-paced, iterative and incremental software development models such as agile: *Release planning* (2.2.2) and *Roadmapping* (2.2.3). Then we reflect upon the perspective that work on software product management in general seems to take regarding agile software development in section 2.2.4 (*Product management literature relates poorly to agile*).

The discussion is continued from the perspective of recent literature on agile software development. In section 2.2.5 (*Out-of-the-box agile relates poorly to product management*) we show that the need to better link long-term product and release planning with agile software development is recognized both in recent practitioner as well as scholarly literature. In section 2.2.6 (*Agile planning with the product backlog and the product vision*) we synthesize existing literature to present how recent practitioner literature perceives how the product backlog and product vision should drive planning in agile software development. The following three sections discuss in detail the identified "cornerstones" of agile planning. Section 2.2.7 examines the *levels of planning* that, according to recent practitioner literature and experience reports can be identified in agile software development.

In sections 2.3.8 and 2.2.9, the planning levels are related to the other two "cornerstones": *progressive refinement of work items* and *work item meta-models*, respectively. Based on the discussion, the notion of *parent-child work item traceability* rises as a possible key for linking product management and agile software development. Sections 2.2.10 and 2.2.11 examine this further in the light of literature on agile software development and goal-oriented requirements engineering, respectively.

In section 2.2.12 we start wrapping up the discussion by presenting one way of how the core product management processes of roadmapping and release planning can, in the light of the discussed literature, be understood in conjunction with using a product backlog, and provide "agile-compatible" definitions thereof. The discussion on linking product management and agile software development is concluded in section 2.2.13 where we examine the implications of the given definitions from the perspective of tool support for managing work items.

2.2.1 The software product management reference framework

Figure 2.1 shows the currently most cited[5] framework that attempts to present an overview of software product management, the *software product management reference framework* (van de Weerd 2006). The software product management reference framework (SPMREF, or simply 'the framework' from hereon) has been developed by studying small and medium Dutch enterprises

[5] Based on searching Scopus for 'software product management' as well as 'product management'; to our knowledge, it is also the only peer-reviewed framework on the topic

(Kittlaus & Clough 2009).

The SPMREF focuses on the engineering aspects[6] of software product management (Kittlaus & Clough 2009) through the four 'core activities' of *portfolio management*[7], *product roadmapping*, *requirements management* and *release planning* (van de Weerd 2006). SPMREF also features the most important internal (e.g. sales & marketing, research) and external stakeholder (e.g. the market, partners, customers) groups (van de Weerd 2006).

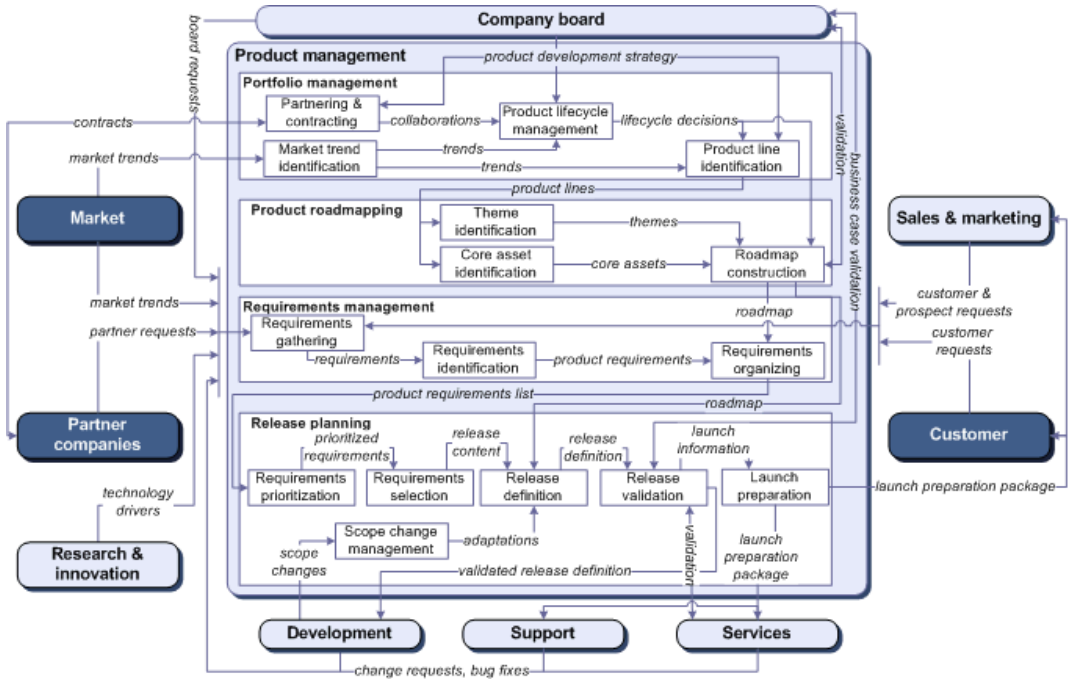


Figure 2.1 Reference framework for software product management (van de Weerd 2006)[8]

The original authors of the SPMREF have recently proposed an updated version of SPMREF, calling it the *software product management competence model* (Bekkers et al. 2010) or SPMCOM for short, to better reflect its use from an industrial perspective. SPMCOM is displayed in Figure 2.2 below.

[6] As opposed to product strategy, pricing, distribution channels, licensing, positioning, messaging and determination of value propositions (Hodgkins & Hohmann 2007, Helferich, Schmid & Herzwurm 2006). Despite of the importance of these non-engineering aspects of product management for long-term success, research indicates that product managers' attention is often spent elsewhere: "Many product managers feel that they would do their job better if they could spend more time on strategy-related efforts and less time on working to expedite projects through other departments" (Murphy & Gorchels 1996).

[7] In this section we adopt a single-product perspective for understanding how agile software development could link with long-term planning. Discussing portfolio management and its link with agile software development will be returned to in section 2.3.

[8] Reproduced here with permission from the original author

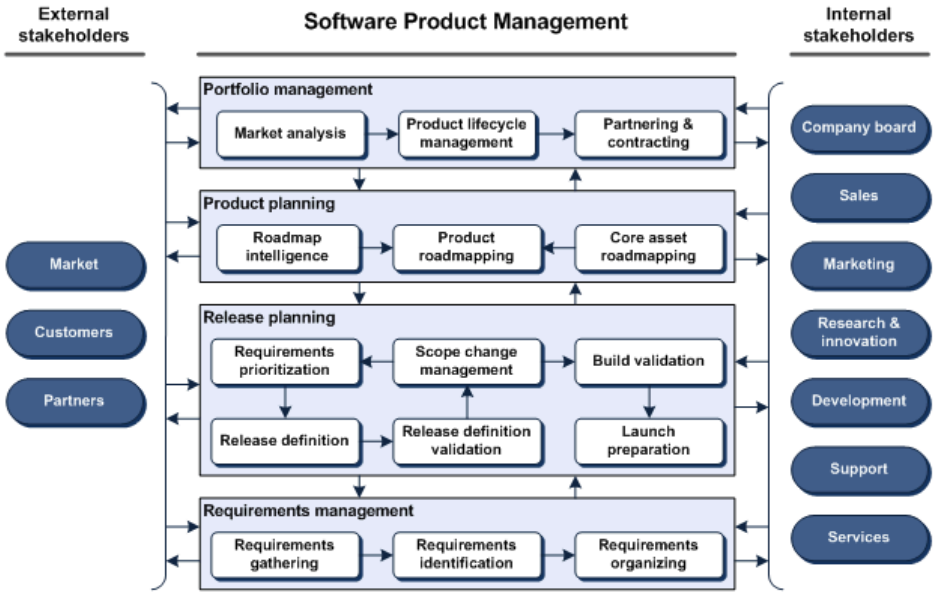


Figure 2.2 The Software product management competence model (Bekkers et al. 2010) [9]

According to Bekkers et al. (2010), the software product management competence model (SPMCOM from hereon) is a re-drawn, and clarified version of SPMREF, with essentially the same contents and message despite the changes in terminology[10].

The following excerpt describes the product and release planning business functions of the competence model (Figure 2.2) central to this study (Bekkers et al. 2010):

> **Product planning** *is focused on the gathering of information for, and creation of a roadmap for a product or product line and its core assets.* […] *Roadmap intelligence gathers decision supporting information needed in the creation of the product roadmap. Product roadmapping deals with the actual creation of the product roadmap itself. Core asset roadmapping concerns the planning of the development of core assets (components that are shared by multiple products).*
>
> **Release planning** *covers the* […] *capabilities needed to successfully create and launch a release. Requirements prioritization prioritizes the identified and organized requirements. Release definition selects the requirements that will be implemented in the next release, based on the prioritization they received in the preceding process. It also creates a release definition based on the selection. Release definition validation is performed before the release is built by the development department. It focuses on the validation of the release definition by internal parties. Scope change management handles the different kinds of scope changes that can occur during the development of a release. Build validation is performed after the release has been realized by the development department. It focuses on validating the built release before it is launched. Launch preparation prepares the internal and external stakeholders for the launch of the new release. Issues ranging from communication, to documentation, training, and*

[9] Reproduced here with permission from the original author

[10] The 'core activities' in SPMREF have been re-termed as 'business functions', their displayed contents are referred in SPMCOM as 'focus areas', and the 'product roadmapping' core activity in SPMREF has been renamed as 'product planning' in SPMCOM. In SPMCOM, the term 'product roadmapping' is used to refer to a specific focus area in the product planning business function.

the preparations for the implementation of the release itself are addressed.

Roadmaps and release plans should be revised at discrete points in time, while *requirements management* is more or less a continuous activity (Kittlaus & Clough 2009).

Both SPMREF and SPMCOM claim to be agnostic with respect to the development life-cycle used (e.g. waterfall, agile, etc.). However, is seems obvious that how product planning/roadmapping (*product roadmapping*, or simply *roadmapping* from hereon) and *release planning* are conducted in practice would play a key role in linking product management and agile software development. Because of this, literature that addresses these processes is examined in more detail in sections 2.2.2 (*Release planning*) and 2.2.3 (*Roadmapping*) below.

2.2.2 Release planning

Release planning (also referred to as "product release planning" and "strategic release planning") is concerned with selection and assignment of requirements in one or more sequences of releases in a way that important business, technical and resource constraints are fulfilled (Svahnberg et al. 2010).

Release planning has attracted attention among software engineering researchers. However, much of the existing academic work, for example (Ngo-The & Ruhe 2009, Al-Emran, Pfahl & Ruhe 2010, Akker et al. 2008, Mc Elroy & Ruhe 2010), treats release planning essentially as an optimization problem (Kittlaus & Clough 2009). According to a recent review by Svahnberg et al. (Svahnberg et al. 2010), the release planning models are designed for a situation where there is a single product/service offering with a set of possible features to be selected from. The models also assume that these features have been elaborated to the degree that their development cost and business value can be reasonably estimated, and a group of relevant stakeholders is readily available to familiarize themselves with the requirements and vote on them.

However, usually one or more of the assumptions listed above does not hold in practice (Svahnberg et al. 2010, Heikkilä, Rautiainen & Jansen 2010, Lehtola & Kauppinen 2006). For example, requirements are usually not prioritised as a one-off activity, but in multiple phases of development, with each phase involving different kind of decision-making (Lehtola 2006). Applying an algorithmic model may also discourage the stakeholders from the direct communication necessary for solving the complex problem of deciding what to include in a release (Ziemer & Calori 2007). Furthermore, the degree of up-front requirements elaboration needed by the optimization models is often not feasible – or even desirable from a lean/agile software development standpoint (Larman & Vodde 2010, Poppendieck & Poppendieck 2009).

Existing systematic algorithmic approaches to planning the future development steps of a particular product/service offering seem to have unfortunately little applicability to the actual decision-making problem faced by practitioners (Ivarsson & Gorschek 2009). Indeed, it could be argued that if the input values needed by the various release planning algorithms could to a sufficient degree be approximated without a considerable amount of up-front planning and negotiations, finding a common decision would be straightforward and thus required no algorithmic models in the first place. If this is so, it is then hardly a surprise that most approaches to release planning have not been validated in an industrial setting (Svahnberg et al. 2010, Heikkilä, Rautiainen & Jansen 2010). And, when it comes to release planning in the context of agile software development, it would seem that expert practitioners of agile are, instead of looking for solutions among stakeholder voting supported by optimization models, in favor of approaches that resemble regular iteration planning but with a larger number of people involved (Heikkilä, Rautiainen & Jansen 2010, Leffingwell

2011)[11].

Thus, rather than further devising models for "optimizing" the contents of upcoming releases, we take the stand that it should first be *conceptually understood how long-term product and release planning actually (are said to) manifest in agile software development*. We continue this discussion in the following section (2.2.3 *Roadmapping*).

2.2.3 Roadmapping

Product roadmapping (or simply roadmapping) is a common metaphor for planning the use of resources, technology and their relationships over a period of time (Kostoff & Schaller 2001). It is a common approach for companies to take in order to bridge the gap between business planning and product development (Lehtola et al. 2009). In general, the process of roadmapping is about identifying, evaluating and selecting strategic alternatives for achieving desired objectives (Kostoff & Schaller 2001), whereas roadmaps summarise and communicate the results of key business decisions (DeGregorio 2000).

The application of roadmapping in the software development has been investigated quite little (Lehtola et al. 2009). Based on the case studies and experience reports discovered in this study, software companies are to some degree aware that project requirements documents are not sufficient to ensure the kind of organization-wide cross-functional understanding regarding the long-term direction of the product (Lehtola et al. 2009). Also, it is not uncommon for product managers create roadmaps in isolation and out-of-synch with both company level strategies as well as development (Lehtola et al. 2009, Wilby 2009). This may partly contribute to personnel viewing roadmaps as "too static" (Lehtola et al. 2009), or at the other extreme, being "modified at whim by countless authors […] but with limited cross-functional collaboration, […] still remaining outdated, inaccurate, and ignorant of dependencies between features and functions in other products" (Wilby 2009). Thus, proper roadmapping seems to be challenging in practice as well.

It is generally agreed that the roadmapping process should be customized to the context as well as an organization's needs (Oliveira & Rozenfeld 2009, Phaal et al. 2003, Fleury et al. 2006). In an attempt to customize roadmapping for software companies, Fleury et al. (2006) explain that a generic roadmap structure should be based on *software development processes*[12] and *software management processes*[13]. However, the article does not present an example roadmap or a tangible roadmap template. Nor does it relate how roadmapping should link with short-term iteration planning practiced in agile software development.

Thus, to provide a foundation for understanding what roadmapping could mean in software development and agile software development in particular, we in the following synthesize what has been said of roadmaps and roadmapping in the context of software product development, agile or otherwise. The synthesis in sections 2.2.3.1-2.2.3.7 below draws both from sources on software

[11] While (Leffingwell 2011) was published thus too late to have been included via the search protocol, we've been following its writing and development since 2009 on the author's web site at http://scalingsoftwareagility.wordpress.com/. Thus, we throughout this dissertation refer to the published book instead of its various in-progress versions such as (Leffingwell 2009) or (Leffingwell & Aalto 2009)

[12] These are: "requirements, design & architecture, programming and tests" (Sommerville 1996)

[13] According to the Software Engineering Body of Knowledge (www.swebok.org) these are: "project management, risk management, quality assurance and configuration management"

product management in general (mostly written by expert practitioners), as well as on agile software development (books written by expert practitioners and experience reports published at conferences). It is structured in terms of the *definition* of a roadmap (2.2.3.1), the *purpose* of roadmapping (2.2.3.2), *what can be included* in a roadmap (2.2.3.3), the *timeframe* for roadmapping (2.2.3.4), how often the roadmaps should be *updated (*2.2.3.5*), who should be involved* in roadmapping (2.2.3.6), as well as the *format of the resulting roadmap* itself (2.2.3.7). This structure was devised by the author as no existing structure for organising a similar discussion was found.

2.2.3.1 What is a "roadmap"

The software product roadmap is a planning artefact showing an overview of how a product is intended to evolve over a strategic timeframe (Kittlaus & Clough 2009).

2.2.3.2 The purpose of roadmapping

Roadmapping is done to give direction both internally and externally (Kittlaus & Clough 2009). However, the process of roadmapping is commonly emphasized over the roadmap itself; that is, planning is more important than the resulting plan (Oliveira & Rozenfeld 2009). Moreover, the roadmaps' implementability is at least as important as any possible strategic value they may, as plans, convey (Kostoff & Schaller 2001).

Internally, roadmaps communicate strategic intent (Hodgkins & Hohmann 2007). In an agile context, the roadmap is said to "facilitate the dialogue between the development team and the stakeholders and coordinate the development and launch of related products, for instance a product line or a product portfolio" (Pichler 2010). The roadmap also guides the prioritization of the product backlog (Hodgkins & Hohmann 2007). Roadmaps can enable the product manager to facilitate the architectural evolution of the offerings, and help the technical team to communicate critical needs and/or opportunities to product managers (Hodgkins & Hohmann 2007). For the product manager, a roadmap may be important for reaching agreement within the company regarding longer-term direction and priorities. The roadmap can also indicate that a product will provide continuing career opportunities for employees who work on it, be they developers, sales people or support specialists (Kittlaus & Clough 2009). Roadmaps can also work to reduce the "person who shouts the loudest gets what they want" effect (Hodgkins & Hohmann 2007).

Externally, the roadmap plays an important role in demonstrating the viability of a product. It provides customers with access to near-term commitments as well as long-term points of view, enabling them to engage in appropriate planning processes (Kittlaus & Clough 2009). Inviting customers to share their perspectives on the future may also further bind them to the company and the offering (Hodgkins & Hohmann 2007). Potential customers may often be willing to sign non-disclosure agreements in order to see a product roadmap before they make a significant investment decision (Kittlaus & Clough 2009). Similarly, market analysts "mostly base their judgment on a convincing story about a product's future as expressed in the roadmap" (Kittlaus & Clough 2009).

2.2.3.3 What can be included in a roadmap

According to the discovered literature, a myriad of things can be included in a software product roadmap – agile or otherwise.

A product roadmap in the context of agile software development should list the upcoming releases, their projected launch dates and (up to) top 5 features (Pichler 2010) with possibly the related business objectives (Smits 2007). The product roadmap should be "simple and focused on the essentials, as the details will emerge and be captured in the product backlog" (Pichler 2010).

To contrast this, a roadmap – again, in the context of agile software development – has been reported to include "everything from planned releases of individual products to the mix of offerings

being targeted to shared market segments, critical market events and market rhythms, moves from competitors, regularly attended business conferences […] thus enabling further agility in product planning" (Hodgkins & Hohmann 2007). This is in line with an experience report by Wilby (2009), who reports a company's roadmapping process to include "a breakdown and segmentation of the relevant markets, happenings in the industry that impact customer business cycles (i.e., events, conferences, competitive milestones), functionality tied to target customers and the benefit it brings them, an explicit examination of the technical infrastructure needed to deliver the features and benefits, a rough estimate of effort required, and mapping these as swim lanes on a loose timeline". Wilby's (2009) roadmap should include a written distribution plan (that is, a list of those who should read the roadmap).

Kittlaus and Clough (2009), while not talking about an agile context, add "financial forecasts and dependencies on other products and technologies" to the mix of things that can be included in a roadmap.

2.2.3.4 The timeframe

The proper timeframe that should be planned for when roadmapping depends on the volatility of the information as well as the planning needs of the impacted internal (marketing, sales, services, financing) or external (clients, vendors, partners) parties (Ferrari 2008). Typically, this timeframe ranges from six months (Kittlaus & Clough 2009, Pichler 2010) up to five years (Kittlaus & Clough 2009). In the experience reports regarding agile software development discovered in this review, the timeframe for roadmaps was up to 18 months (Wilby 2009, Hodgkins & Hohmann 2007, Smits 2007).

In general, the roadmap can be detailed and precise for the short-term timeframe, but the more it looks into the future, the less precise it tends to be (Kittlaus & Clough 2009). Only the immediate future – "the first one to two years" according to Kittlaus and Clough (2009) as well as Ferrari (2008), and "the next 6-12 months rather than next two to three years" according to Pichler (2010) – in a roadmap is "more or less reliable, though still subject to slippages caused by development" (Kittlaus & Clough 2009) or changes in direction (Pichler 2010).

The product roadmap should cover a realistic planning horizon and "crafting a product roadmap that covers the next three years provides little benefit" (Pichler 2010) – except, of course, for serving as a formal way of communicating long-term commitment to the product (Kittlaus & Clough 2009).

2.2.3.5 Update frequency

In the context of agile software development, product roadmaps "are living documents, and evolve and change" (Pichler 2010), and they should be reviewed and updated quarterly (Wilby 2009). According to Pichler (2010), a product roadmap should be created once the product has been successfully introduced into the marketplace.

Kittlaus and Clough (2009) write that the updating the roadmap "is usually updated as part of the corporate planning cycle".

2.2.3.6 Who should be involved in roadmapping?

Roadmapping is a highly collaborative process (Wilby 2009). It should be performed by a cross-functional team (Oliveira & Rozenfeld 2009) and supported by an influential stakeholder (Wilby 2009). According to Pichler (2010), the relevant people to create and update the product roadmap when using Scrum include the development team and the product owner, with the person in charge of the product portfolio and representatives from other product development teams possibly also involved. Hodgkins and Hohmann (2007) report that the parties "responsible for roadmapping were product managers and senior technical architects". According to Smits (2007), "creation of the

roadmap is largely driven by the product owner or product owner team".

To summarize, according to the discovered literature, roadmapping should be explicitly sponsored by top management, and performed by a cross-functional team, involving sales & marketing, product management and development.

2.2.3.7 Format

In general, roadmaps are advised to be written down and communicated in a visual format. A visual roadmap should employ symbols the audience can already understand (Oliveira & Rozenfeld 2009). This speaks in favour of a standardized roadmap notation throughout an organization. Also, in the case of having roadmaps for multiple product/service offerings, the use of an uniform notation has been reported to make the creation of "portfolio level roadmap at the business unit level [based on the individual product roadmaps] significantly easier" (Hodgkins & Hohmann 2007).

However, there seems to be no easy answers for whether the visual roadmap should be a physical or an electronic one. According to Smits (2007), the roadmap is literally drawn by the hand of the product owner. Wilby (2009) relates an experience where the roadmap is initially created on a wall with post-it notes and twine. However, because of the need to share the results to a broader audience (and after "much trial and error with digital photographs"), the final roadmap was redrawn with a graphical tool "so that it could be easily distributed"[14].

2.2.4 Product management literature relates poorly to agile

In a manner similar to the discussed software product management reference framework (Weerd et al. 2006b) and competence model (Bekkers et al. 2010) discussed in section 2.2.1, much of the literature on software product management discovered in this review seems to view development as an activity that can be planned for and then carried out according to the plan[15]. For example, in Figure 2.1 (*Reference framework for software product management* (van de Weerd 2006), page 10), scope changes from development link to requirements management, product roadmapping and portfolio management only indirectly. In this section following we present several excerpts that seem to support the notion that at least much of the literature on software product management has been written with a sequential, waterfall-like software development life cycle in mind. If this is the case, existing literature would offer little help for linking product management with agile software development.

In one of the earliest books on software product management by Condon (2002), several pages are dedicated to discussing eXtreme Programming. The following excerpt hints that the fundamental concepts behind agile software development may be foreign to the more *plan-driven* (Boehm & Turner 2003) worldview that seems to have been adopted by many authors on software product management:

> *There are many detractors of eXtreme Programming.* [The reasons] *stem primarily from some of its unorthodox requirements, such as having people from different disciplines sit together or incorporating QA tasks right back into engineering. These aspects* […] *have created a fair amount of resistance* [which XP would have not met if the changes were isolated into the software development process only]. *Let's look first at some of the problems XP may cause, and then discuss ways it may still be adopted.* (p. 84)

[14] Wilby (2009) does not mention whether the next versions roadmap(s) went through the same transformation cycle (first post-its and twine, then re-drawn with a graphical tool) as well.

[15] However, it is recognized that "slippages caused by the development" can happen (Kittlaus & Clough 2009), page 77.

This mindset seems to be present also in more recent practitioner-oriented books that have been published after the rise of agile software development as part of mainstream software engineering. For example, the following excerpts from (Kittlaus & Clough 2009) concerning the documentation of requirements seem to be written from a plan-driven perspective:

> *Requirements management is a documentation-intensive task. Each requirement must be individually documented. Then all requirements from all the diverse sources are combined into a one document, the high-level specification of a product (release). This is the main working document of product requirements management. From this the technical specification document is derived.* (p. 90)

> *When requirements management has documented the requirements in this way, and the product manager has categorized, evaluated and packaged them into a release, he will combine the release requirements in a high-level specification.* […] *This high-level specification for a product (release) is the basis for development and for the technical specification which has a technical perspective compared to the customer perspective of the high-level specification.* (p. 91)

> *The described documents aim at a common understanding of all parties involved. This must not result in a process that is too inflexible. The project requirements management must allow changes during the development process in a tightly managed manner.* (p. 92)

The plan-driven mentality exhibited above seems in stark contrast to the continuous planning and just-in-time elaboration (Shalloway, Beaver & Trott 2009) advocated by the agile movement (see sections 2.2.6-2.2.9 below).

As another example, the following paragraph contains Kittlaus and Clough's software product management book's (2009) sole discussion on agile software development:

> *Changes to requirements during the development process are a frequent source of conflict. From a development perspective, it would be ideal if all requirements were completely and precisely defined at the beginning of a project without any subsequent changes during the project.* […] *Since the good old waterfall model that assumes a strictly sequential process cannot cope well with late changes, newer approaches like incremental models or Extreme Programming that assume iterations are better suited to handling change. Their disadvantage is that it is more difficult for the software product manager as contract giver to evaluate the status of a project than with the waterfall model.* (p. 92)

Judging from the above examples, we suspect that both practitioner as well as scholarly literature on software product management may currently be disconnected from agile software development. The following excerpt from a recent practitioner book on lean requirements engineering in software development by Leffingwell (2011) seems to address the underlying problem:

> *To those […] in the software product management and product marketing community, this [the role of product owner in agile methods] looks like a new version of "barbarians at the gate", whereby the development community is attempting to extend their control into areas where they lack competence, background, and training. To them, Scrum is a process, built by developers, for developers. Who says that these processes new roles will now be used to drive product definition and policy?* (Leffingwell 2011), p. 204

This seems to match our observation that software product management practitioner and research communities are only just beginning to connect their worldview with that of agile software development community. First steps are in fact being taken, but so far, the approach has been that of "applying agile principles to product management" (Vlaanderen et al. 2009), rather than figuring out how product management should be conducted to work in unison with an agile software development process.

In fact, based on the articles found in this literature review, the above mentioned approach of "applying agile principles" may be more widespread in contemporary scholarly software engineering literature. The implications of agile software development methods seem to be first to

examined in terms of *making parts of the from-idea-to-launch value stream more agile*, as represented by different research communities; for example, requirements engineering – market-driven (Dzamashvili-Fogelström et al. 2010) or in general (Sillitti & Succi 2005) – product line engineering (Mohan, Ramesh & Sugumaran 2010), or in this case, software product management (Vlaanderen et al. 2009). While all of these steps certainly are in the right direction and pave the way forward, it may, from the perspective of enabling the entire organization to benefit from agile software development methods, be more beneficial figure out a proper conceptual integration between agile software development and product management core areas such as roadmapping and release planning.

With this in mind, we in the following sections continue the discussion on linking product management and agile software development from the perspective of practitioner as well as scholarly[16] literature on agile software development. First, we examine whether the need to improve linking product management and agile development is recognized in the literature on agile software development (section 2.2.5). The sections that follow provide an in-depth discussion of how the product backlog and product vision are said to drive planning in agile software development (section 2.2.6), including the related subtleties, such as levels of planning (section 2.2.7), progressive refinement of work items (section 2.2.8), work item meta-models (section 2.2.9) and work item parent-child traceability (sections 2.2.10 and 2.2.11). Based on the discussion, we in section 2.2.12 propose that one way of conceptually linking roadmapping and release planning with agile software development is to augment the concept of product backlog – commonly thought of as a "flat list" – with an explicit work item hierarchy, and provide "agile-compatible" definitions of roadmapping and release planning. Section 2.2.13 concludes the discussion on linking product management and agile software development by discussing the implications of our definitions for work item management tool support.

2.2.5 Out-of-the-box agile relates poorly to product management

In Scrum, the entire area addressed by the software product management reference framework (see section 2.2.1) is supposed to be handled by the so-called product owner role, who should be a single person (Schwaber & Beedle 2002, Schwaber 2007, Schwaber & Sutherland 2010). Some authors, however, view that in many situations, product ownership has to be tackled by at least two people (Leffingwell 2011), an experienced group (Shalloway, Beaver & Trott 2009, Pichler 2010, Galen 2009), a steering committee (Ktata & Levesque 2009) or an appropriate hierarchy of product owners (Woodward, Surdek & Ganis 2010, Keith 2010).

While the bottom line of this discussion seems to be that there should be a single, team-facing product owner who then may or may not have multiple teams – depending on the situation (Larman & Vodde 2010), it is certainly true that agile software development attempts to hide much of the complexities of product and requirements management into simply *using a product backlog,* with the product owner being responsible for dealing with the complexity (Cao & Ramesh 2008, Paetsch, Eberlein & Maurer 2003). Accordingly, product ownership and using the product backlog are arguably the most challenging concepts in agile software development. Based on the literature discovered in this review, a pattern seems to emerge: if fast-paced development producing small increments is not properly managed, the contribution of the fragmented results to the projects' goals – or even company's overall strategy – can easily be lost. The following paragraphs discuss this in more detail.

[16] "Scholarly" being a relative term, as most of the scholarly literature on agile software development actually consists of experience reports published at conferences.

When using the product backlog, complex requirements are supposed to be split into smaller requirements (Ambler 2008b) in order to be easier to estimate, implement, and validate (Ambler 2008c). However, keeping track of the big picture becomes more difficult (Ambler 2008c), and the resulting smaller requirements may not have value outside their original context (Ambler 2008b).

Kalliney (2009) reports "individual user stories being so small" that it became difficult for business owners and product managers to validate that product vision or enterprise strategy were being properly implemented. Sprint review demos emphasized the work that was accomplished during the given sprint and rarely addressed the product vision (Kalliney 2009). The metrics provided ("burn-ups or burndowns") did not help in assessing whether the delivered functionality was properly aligned with the product vision (Kalliney 2009). Likewise, at Nokia, management was concerned that after the transition to agile, product backlogs and agile planning would only provide short-term visibility (Laanti 2008).

Lehto and Rautiainen (2009) conducted a case study of software development governance challenges in a medium-sized company. In the described case, the development teams had "abandoned task-level planning", which "made it impossible to monitor progress on a detailed level". Also, even if the progress of individual sprints could be monitored, the lack of explicit linking between the sprints' work items and the higher level goals made it impossible to link development progress to high-level plans. Consequently, no true progress information was available, making it impossible for management to take corrective actions in time.

According to Vidgen and Wang (2009), agile software development requires both high-level, sketchy, long-term plans and detailed, accurate, short-term plans. In line with this thinking, Hodgkins and Hohmann (2007) view the product backlog as "a powerful mechanism for managing the efforts of the development team", but "insufficient to address the strategic management needs of most software companies":

> *Unlike the product backlog, roadmaps are intended to identify and clarify the strategic intent of your product. […] Perhaps the most important benefit is that roadmaps provide a point of view about the future that is critical for successful agile development. Backlogs are great, and we need them, but you can't get a sense of where you're headed or why this destination is important just from looking at the backlog. A roadmap, on the other hand, provides exactly this, in a format that promotes communication, collaboration, and understanding on the markets being served, the key features and benefits being offered to these markets, and the necessary technical infrastructure required to realize these benefits.* (Hodgkins & Hohmann 2007)

However, roadmapping may provide little benefit to guide agile software development, unless care is taken to integrate the two processes. Hodgkins and Hohmann (2007) write of being "continually surprised" that agile teams do not practice roadmapping, and suspect that "roadmapping, like other forms of strategic thinking, is challenging in many agile-centric contexts because the roadmaps' longer timeframes are "at odds with the short term horizons of agile methods". In line with this view, Wilby (2009) reports that "it was hard to use the existing roadmap to guide backlog prioritization in the absence of higher level 'epic' stories". Not only did the existing roadmap lack detail needed to provide an overall theme and direction, but it was outdated, inaccurate, and did not utilize the flexibility that agile software development would have made possible. At first, the roadmaps were discarded altogether and it was assumed that the product backlog and user stories were sufficient to provide the direction the development teams needed. However, without roadmaps, "agile began to feel very chaotic":

> *Those intimately involved with a project were happy* […]*, but for everybody else there was a great sense of confusion. Without the roadmap as a reference point it was extremely difficult to interpret the work being done in engineering. Even attending stand-ups or sprint reviews did not ease the growing sentiment that development was just working on whatever they wanted. For departments outside of product management and engineering, such as*

> *marketing or sales, a sense of mistrust began to grow, because when they asked even the simplest of questions around what was going to be in the next release, the answer was increasingly "that depends."* […] *Sales started to complain that they could not answer customer and prospect queries around the overall future of products and services without a committed roadmap being available.* [These factors were] *beginning to put the value of Agile to the overall company business into question. The lack of product roadmaps was being cited at the highest levels as evidence that Agile wasn't working for us.* (Wilby 2009), page 2

According to Wilby (2009), the transition to agile software development eventually forced "the skeletons in the closet [regarding the roadmapping process and the roadmaps] out into the daylight". Roadmapping was taken up again, with the engineers and architects now playing an important role in the roadmapping process. Also, with the new roadmaps, development grew more confident as the user stories could be traced to the customer benefits, target customers and market rhythms indicated in the roadmaps (Wilby 2009). Likewise, according to Hodgkins and Hohmann (2007) "the challenges [of integrating roadmapping with agile software development] can be overcome, […] once a base roadmap is in place it can be managed using agile-centric techniques of responding to change over following a plan."

According to some researchers, the reported experiences discussed above are a natural consequence of the origins of agile software development. Agile software development originally adheres to the viewpoint of a single development team dedicated to an individual project (Kettunen & Laanti 2006, Racheva, Daneva & Buglione 2008). Because of this single-project focus, agile methods lack support for long-term product development and they do little to address how the achievements of individual development projects benefit the short- and long-term business goals of the organization (Dzamashvili-Fogelström et al. 2010). Furthermore, agile methods "may have a detrimental effect on long-term product development […] by disabling effective product management" (Dzamashvili-Fogelström et al. 2010). Overall, if the benefits purported of agile methods are to be realized for the entire organization (Laanti 2008), agile methods should be extended from addressing individual projects to "longer-term product evolution and portfolio management" (Kettunen & Laanti 2006).

While extending agile methods to link with product and portfolio management may not be simple, it can be an extremely worthwhile pursuit. For example, the most important challenges that in general face practitioners in the context of market-driven software development have been identified as marketing-to-development communication, requirements traceability, release planning, and finding the time to produce detailed requirements documentation (Karlsson et al. 2007). These challenges are something that agile methods can also be considered to suit especially well – despite of the problems mentioned earlier (Mishra & Mishra 2009). Also, traditional methods may be even less suited for market-driven software development than agile software development (Petersen & Wohlin 2010). From this perspective, the above discussed observations made by Dzamashvili-Fogelström et al. (2010) are likely to reflect the pains involved in the significant cultural transformation that a large-scale agile transtition is bound to cause (Moe, Dingsyr & Kvangardsnes 2009). If so, they also exemplify how difficult applying agile software development "out-of-the-box" can be for both practitioners and researchers alike.

Guidance for extending agile methods to link with product and portfolio management has in the recent years started to emerge in books written by expert practitioners and consultants and in industrial experience reports published at both practitioner as well as scholarly conferences. We discuss this work further in the sections that follow.(Petersen & Wohlin 2010)

2.2.6 Agile planning with the product backlog and the product vision

In this section we synthesize existing literature[17] to describe how the product backlog and the product vision should drive the planning of development efforts in agile software development. The context discussed is mostly that of the Scrum framework (Schwaber & Sutherland 2010). This is because Scrum is the most widespread agile framework (Krebs 2008), management-focused and to some degree agnostic of the technical practices used (Mohan, Ramesh & Sugumaran 2010). Thus, it is also the agile software development framework best suited for this kind of discussion.

The *product vision* drives the prioritization of the product backlog by describing the long-term objective of the product (Schwaber 2007) and a high-level view of the business opportunities involved (Woodward, Surdek & Ganis 2010). Vision should be reflected upon not only in release planning, but in sprint planning and daily conversations as well (Keith 2010).

The *product backlog* is a prioritized list of all the work that currently can be seen as useful to perform in order for offering, either new or already existing, to succeed and prosper (Schwaber 2007). It contains all the features, functions, technologies, enhancements and bug fixes that constitute the changes that could be made to the product for future releases (Pichler 2010). In addition, the items on the product backlog (referred to as *work items,* or simply *items* from hereon) can also be non-functional, such as reviewing the output of another team, attending a training session, setting up equipment (Ambler 2008b), constraints (Ambler 2008a), architectural enhancements (Hodgkins & Hohmann 2007) or work that needs to be done to remove technical debt (Schiel 2009). The work items in the product backlog are ordered sequentially according to priority, with the topmost items being viewed as more important and/or urgent than those beneath them (Schwaber 2007).

While the product backlog is constantly emerging and changing (Schwaber 2007), the ideal is to have at least the top of the backlog constantly rank-ordered, up-to-date and ready to feed development teams with detailed-enough user stories (Cohn 2009). The higher the items are in the product backlog, the clearer and more detailed they should be in terms of their description and effort estimates.

The product owner should, as development proceeds, continually work on the product backlog with the help of the development team (Schiel 2009). In this progressive refinement and re-prioritization of work items, also known as *backlog grooming* (Schiel 2009), larger, vague work items that reside lower in the product backlog (Keith 2010) are split into smaller and more detailed work items (Cohn 2009).

To avoid giving the impression of precise estimates for large work items, it is often recommended that work items further down the backlog should be estimated in abstract terms that have no clear connection to calendar time, for example story points or T-shirt sizes (e.g. XS, S, M, L, XL, XXL) (Schiel 2009, Cohn 2005). However, some authors still recommend that "ideal days" should be used (Hodgkins & Hohmann 2007, Stober & Hansmann 2010, Tengshe & Noble 2007) to make the normalization of estimates possible for various reasons.

The ideas concerning the product backlog mentioned in the above paragraphs are illustrated in Figure 2.3 below.

[17] The work referenced in this as well as the following sections mostly consists of recent practitioner books and industrial experience reports published at conferences.

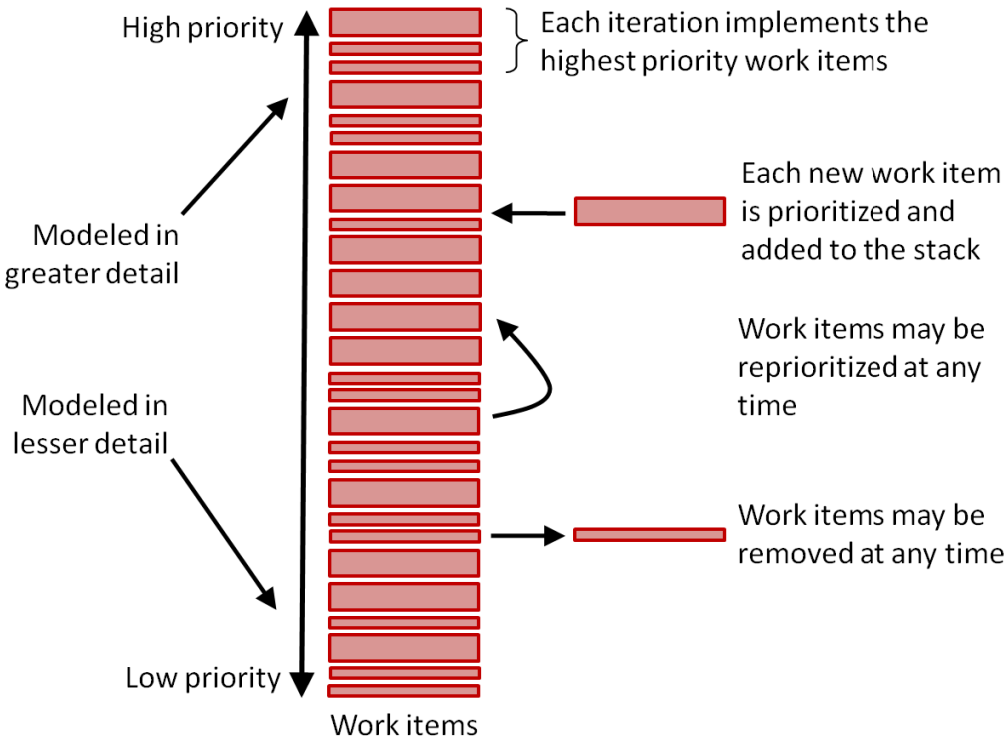


***Figure 2.3 The product backlog as a prioritized, constantly emerging list of work* (Ambler 2008b)**

Because the product backlog transcends any single release (Schwaber 2007) it can be viewed as consisting of several sections (Schiel 2009). The topmost section, *sprint backlog*, contains those work items that have been committed to for the ongoing development iteration, including the tasks that are needed to get the work items done (Schwaber 2007). While work items should be estimated in story points (or ideal days, or T-shirt sizes, etc), tasks are generally recommended to be estimated in man-hours. Also, up to 40% of the tasks might emerge during the sprint (Schwaber 2007). In the case of multiple teams working in parallel on the same product, each team has its own sprint backlog (Woodward, Surdek & Ganis 2010).

Beneath the sprint cut-off line resides a section called *up next* (Schiel 2009). In this section, all of the items are undergoing final preparation and are split into stories that are so small that they can be easily handled by a development team. This is where the majority of backlog grooming occurs in terms of effort spent in team/product owner –co-operation (Schiel 2009).

Beneath the up *next section* reside the rest of those work items that are currently thought as necessary to get done before the version of the product currently under work is launched (Keith 2010). These work items have been reviewed at least at the start of the ongoing release project, with some of the items split into a reasonable (but still relatively large) size in order to get a rough understanding of the effort required to put the release together (Schiel 2009). Together with the contents of the current sprint and the items up next, this comprises the current release plan.

Those items that currently are seen as being worked on in future releases reside beneath the cut-off line of the current release (Schiel 2009, Cohn 2009). For new items, no analysis has been

completed, nor is any planned (Schiel 2009). Besides those work items that have been planned to be included in some future release, the product backlog may also contain work items that have been thought of, but so far have not been seen as crucial enough to attend to in any of the future releases (Schiel 2009). These reside in the *someday* section.

Despite the all-inclusive nature of the product backlog, it should not "become a parking lot for every possible feature that could potentially be in the product" (Woodward, Surdek & Ganis 2010). Indeed, practitioner authors commonly recommend that a possible idea pool should be kept separate from the backlog itself (Schiel 2009).

With the exception of an idea pool, the concepts mentioned in the above paragraphs are illustrated in Figure 2.4 below adapted from (Schiel 2009). The lengths of the red bars indicate work item size.

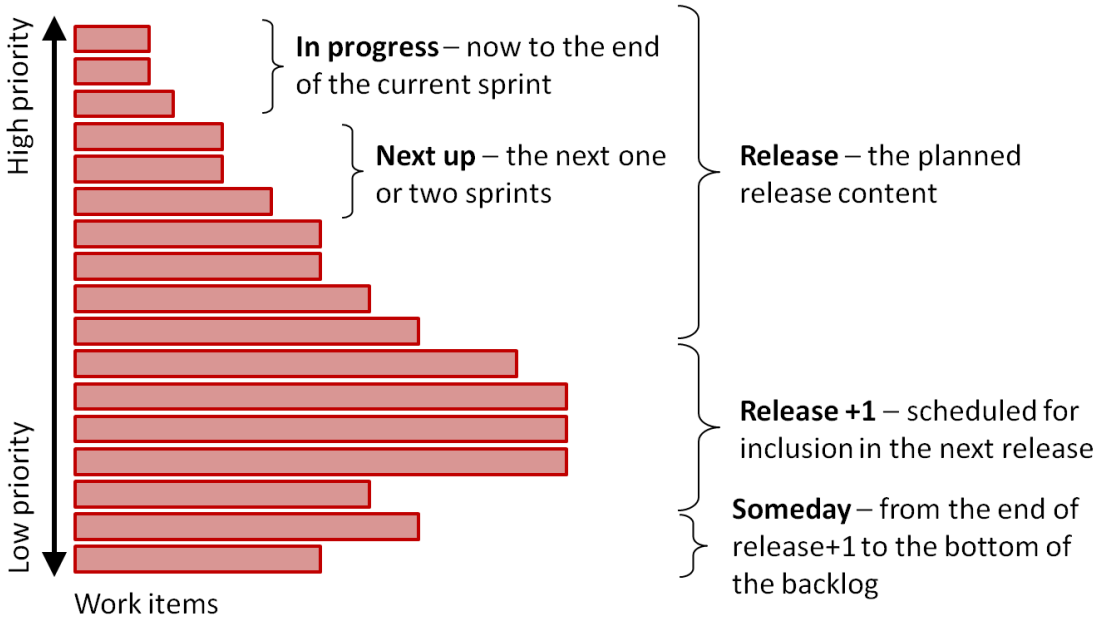


***Figure 2.4 Sections in the product backlog, adapted from* (Schiel 2009)**

While Figure 2.3 and Figure 2.4 above depict the product backlog as a strictly sequentially ordered list, only the rank-order of the topmost items can in practice be considered to be 'thought through' and absolute (and even this is very much subject to change). The work items committed to for the sprint should be in absolute rank-order. However, the items further down the backlog – especially beyond the *up next* section – need not to be prioritized with the same accuracy (Keith 2010). The product planning iceberg in Figure 2.5 below depicts the inherent uncertainty in work item priorities further down the backlog by placing more than a single item vertically on the same level. The areas of the red bars indicate work item size.

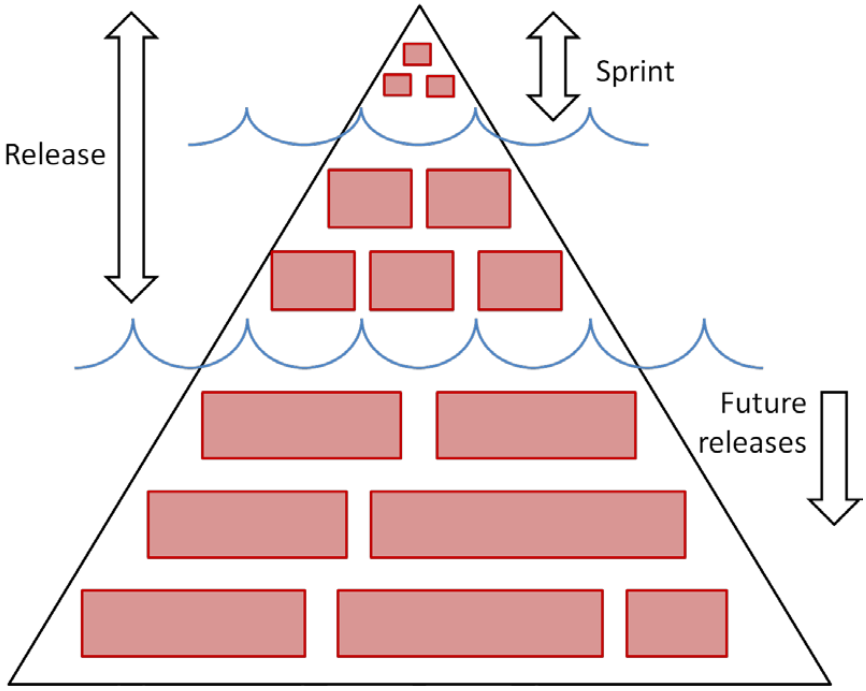


***Figure 2.5 The product planning iceberg, adapted from* (Keith 2010)**[18]

Also, while this is not illustrated by any of the above figures – as we could not find an existing illustration depicting this property – the lower sections of the product backlog may, in addition to the large and vague work items, also contain items that are small and detailed (Schiel 2009). An example of such an item could be a laborious but basically simple and well understood bug fix that has not been seen as crucial to attend to yet.

With the exception of the product vision and its role in driving the planning, we have summarized the properties of the product backlog discussed so far into Figure 2.6 (*The product planning volcano*) below.

[18] Figure 2.5 has been adapted from Figure 6.1 in (Keith 2010), p. 111. Clinton notes that the iceberg metaphor has first been presented by Mike Cohn. However, apart from various presentation slides at Cohn's home page (http://www.mountaingoatsoftware.com/), it has to our knowledge only been published in (Cohn 2009). As the version published in Cohn's book (2009) omits the lines separating the sections of the product backlog – which are essential from the point of view of this study, and do appear in Cohn's presentation slides – we have here chosen to cite Clinton's book instead.

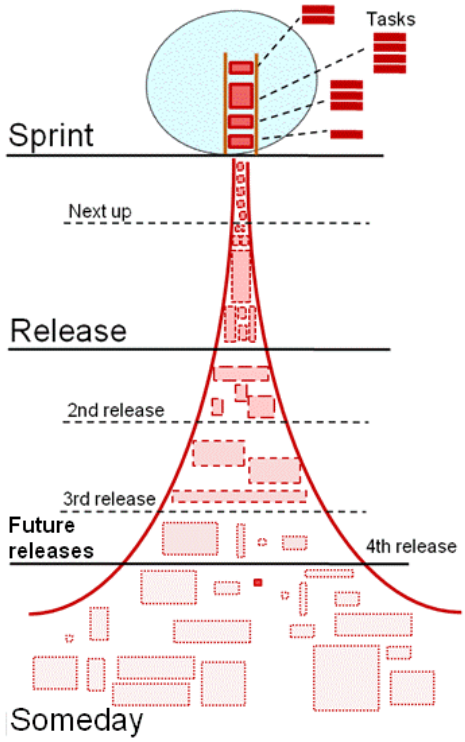


Figure 2.6 The product planning volcano

In Figure 2.6, the areas of the red bars indicate work item size[19]. We will return to discuss our summary of the properties of the product backlog illustrated above in Figure 2.6 in section 2.2.12 (*Linking release planning, roadmapping and the product* backlog) on page 34. But before this, we in the following sections discuss some of the subtleties implied by the product backlog concept in more detail. These are *levels of planning* in agile software development (section 2.2.7), *progressive refinement of work items* (section 2.2.8), *work item meta-models* (section 2.2.9) and *parent-child work item traceability* (sections 2.2.10 and 2.2.11). As we will explain, these listed subtleties form the building blocks for conceptually linking release planning and roadmapping with the product backlog.

2.2.7 Levels of planning in agile software development

As discussed in section 2.2.5 (*Out-of-the-box agile relates poorly to product management*), planning only a single iteration at a time may only be sufficient in the case of small projects. If this is done for projects that have multiple teams or run for more than a few iterations, a "view of the whole" may easily be lost (Smits 2007).

Looking at the literature on agile software development discovered in this review, instead of separate *phases* for planning, several *levels of planning* relevant for agile software development emerge. These planning levels range from the revision of company level goals to the task-level

[19] Note, that the topmost, sprint section is a 'zoom-in'; thus, the work items there items are meant to be roughly of the same size as those in the 'next up' section.

coordination and synchronization that occurs in the daily meetings of a development team. While our review discovered several models of planning levels in agile software development (Lehto & Rautiainen 2009, Hodgkins & Hohmann 2007, Smits 2007, Rautiainen, Lassenius & Sulonen 2002), these are essentially variations of the "planning onion" concept presented by Mike Cohn (Cohn 2005)[20]. The planning onion describes the time horizons essential for planning agile software development, and can thus be used to help connect agile software development with product management, as expressed by e.g. the software product management reference framework (Weerd et al. 2006b) and competence model (Bekkers et al. 2010) discussed in section 2.2.1.

In Figure 2.7 below, we have complemented Cohn's planning onion (2005) with the planning artifacts and roles from Smits' (2007) model of agile planning levels.

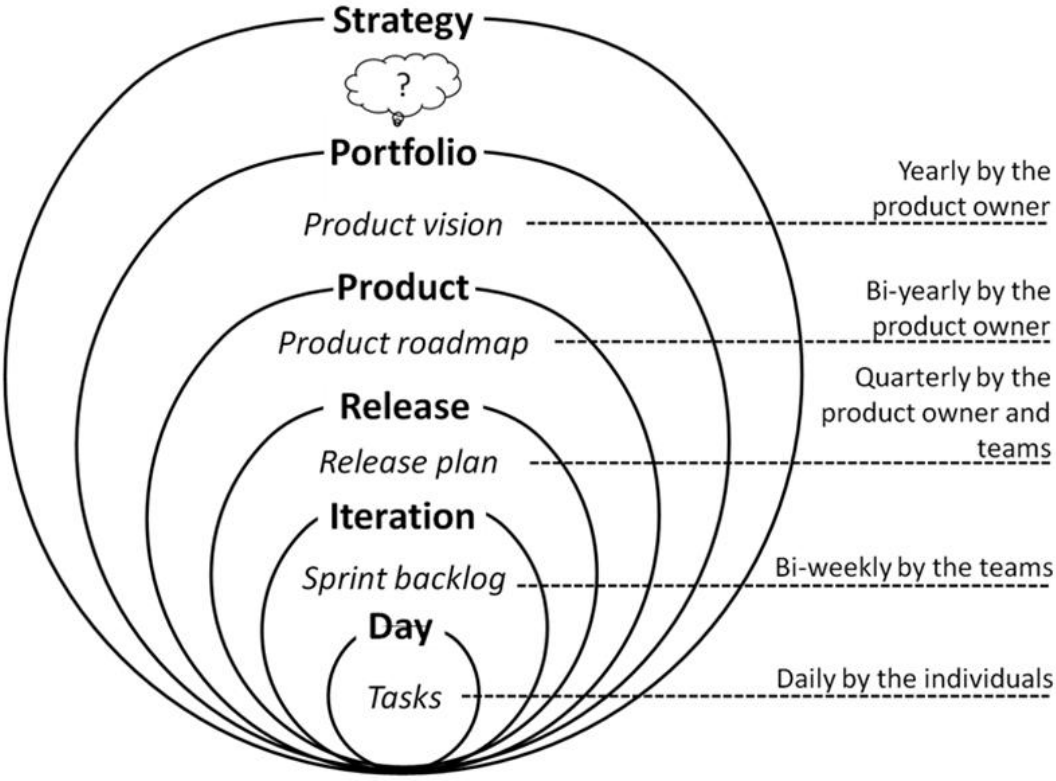


Figure 2.7 Planning levels, artifacts and roles in agile SW development;
(Smits 2007, Cohn 2005)

[20] Instead of the Cycles of Control framework (Rautiainen, Lassenius & Sulonen 2002), we have chosen in this literature review to use Mike Cohn's planning onion to link our work with that of other authors. Cohn's planning onion and the Cycles of Control framework presented in publication I have, to the best of our knowledge, emerged independently. This may be true to the other mentioned planning level frameworks as well, although we have little knowledge of their origins. Likewise, the author of this dissertation has first discussed planning artifacts and the roles involved in a manner similar to Smits (2007) even before publication I in (Vähäniitty 2004a). Thus, Figure 2.7 is our contribution, its components have been presented by other authors as well.

The levels of planning depicted in Figure 2.7 are *Strategy*, *Portfolio*, *Product*, *Release*, *Iteration* and *Day*. As can be seen from Figure 2.7, the highest level, Strategy, is mostly omitted in the descriptions of the found planning level frameworks. Portfolio planning involves the selection of the products that will best implement a vision established through an organization's strategic planning[21] (Smits 2007).

In product planning, the product owner looks (at least) twice a year beyond the ongoing release, updating the product vision and revising the product roadmap (Smits 2007). The product roadmap summarises and communicates the contents of several releases at a high level (Lehto & Rautiainen 2009, Woodward, Surdek & Ganis 2010).

Release planning – recommended to be done at least quarterly by the product owner and the teams (Smits 2007) – considers those work items that will be included in the next release in order to determine an appropriate answer in terms of scope, schedule and resources (Cohn 2005). Release planning is not an isolated effort, but occurs throughout a project so that the release plan always reflects the current expectations about what will be included (Cohn 2005). A typical release will span from two to nine months, with three to four months being most common (Smits 2007, Cohn 2005).

In iteration planning, the product owner identifies what the team should address in the new iteration based on the work done so far (Cohn 2005). Iteration planning should also result in spelling out and estimating the tasks that are needed to transform the work items into action, most commonly, working and tested software (Cohn 2005). To help the team align with long-term goals, it is crucial for the product owner to discuss how the sprint contents contribute to the product vision (Woodward, Surdek & Ganis 2010). A common recommendation for iteration length seems to be two weeks, e.g. (Pichler 2010, Smits 2007).

The shortest planning timeframe occurs on the level of daily work via some form of daily stand-up meeting to coordinate work and synchronize daily efforts. Here, the focus is on coordinating the individual actions that lead up to the completion of tasks and subsequently, the corresponding work items (Cohn 2005, Rautiainen, Lassenius & Sulonen 2002).

2.2.8 Levels of planning and progressive refinement of work items

The notion of planning levels in agile software development (section 2.2.7) implies *progressive refinement of work items* (Schiel 2009). As development proceeds, the bigger, vague work items are split into smaller and smaller – according to the planning horizon in question (Cohn 2004). Large items that can be seen as composed of smaller items that have "mixed priority" (Cohn 2005) should, when possible, be split to "get just the bit that adds [the most] value" (Beck & Fowler 2001) so that only that the most valuable "bits" can be developed (Cohn 2005). An example of this is illustrated in Figure 2.8 below.

[21] Note, that while this corresponds to the notion of "portfolio management" as used in the Software Product Management Reference Framework (Weerd et al. 2006a) in section 2.2.1, it covers – as will be discussed in section 2.3 – only a subset of the range of portfolio management decision-making relevant to agile software development.

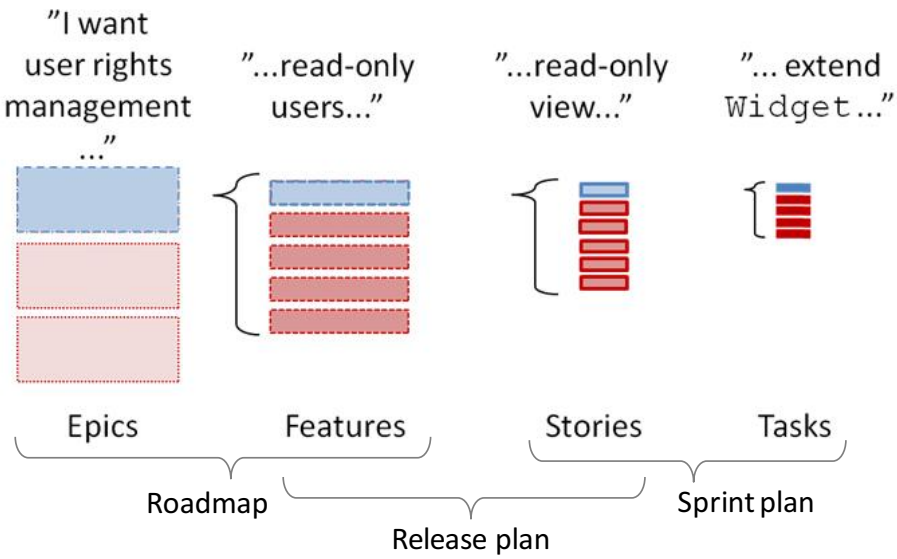


Figure 2.8 Progressive refinement of work items to get ‘the bit that adds the most value’

Figure 2.8 presents an example of progressive refinement of work items concerning a fictitious backlog management tool. The example has been devised by the author and is based on the reviewed literature (see the following paragraphs). The red bars denote requirements (expressed as user stories), the area of a red bar denotes the estimated effort needed to get the user story ‘Done’, and the changes in color depth indicate how the level of detail in the requirement (for example, light red = less detailed). As depicted in Figure 2.8, the progressive refinement of an initial, epic-size user story reveals that the team only needs to develop a new type of “widget” to let a part of a particular product backlog to be published on a web page. This way, the actual need behind the Epic-sized request of extensive user rights management functionality can be met with spending only a small amount of effort, as the widget can be used to let selected ‘outsiders’ view progress directly from the work item management system.

The splitting on the upper levels (that is, in roadmapping and release planning) continues until there are enough small items to “fill” the level below. The refinement of these work items is then continued, and finally paused when the resulting items are small enough so that, according to the estimates, at least 3-4 items can be built in an iteration by a single development team (Beck & Fowler 2001, Larman & Vodde 2008). As an example (see Figure 2.8), Roadmapped Epics are refined into Features until the Release plan fills up; Features are refined into Stories until there is at least one Sprint worth of Stories, and so on.

Progressive refinement of work items should be done just-in-time and according to work item priority (Keith 2010, Larman & Vodde 2008, Cohen 2010). The size of the “clear-fine subset of the release backlog” should be limited (Keith 2010, Larman & Vodde 2008, Cohen 2010). Larman & Vodde (2008) state that a long list of fine-grained complex features is both hard to prioritize and understand, and report of regularly hearing complaints that the backlog “is too big for the product owner team to get their head around”. Cohen (2010) sees work item elaboration as a matter of balance: larger work items are “easier to manage”, while smaller items are “easier to feed into development”.

As a work item in agile software development is essentially thought of as “nothing more than an

agreement that the customer and developers will talk about the feature" (Beck & Fowler 2001), limiting elaboration to only those items that are taken on in the immediate future helps prevent unnecessary work, for example in the form of generating information via analysis that becomes obsolete or is forgotten (Larman & Vodde 2008). A large backlog of clearly analyzed, finely split and well estimated user stories can be seen as an inventory with no return-on-investment, creates financial risk (Larman & Vodde 2008) and breeds a false sense of control (Keith 2010).

2.2.9 Progressive refinement of work items and work item meta-models

In order to support progressive refinement of backlog items, some practitioner-authors on agile software development have in their recent books (Larman & Vodde 2010, Galen 2009, Keith 2010, Cohen 2010) recommended that an organization should establish terms for denoting work items of different size – and thus, abstraction level.

This recommendation is in line with results from requirements engineering research as well. In 2006, Gorschek et al. – studying market-driven software development – coined the term *requirements abstraction model* to refer to a meta-model that explicates both the level of planning as well as the requirements' level of abstraction (Gorschek & Wohlin 2006). They propose that elicited requirements should be organized according to their abstraction level, with parent-child links made explicit for all requirements, even when this requires the creation of new, higher level parent requirements (Gorschek & Wohlin 2006).

The meta-model of Gorschek et al. (2006) defines four abstraction levels: *goal*, *feature*, *function* and *component.* However, the exact number of abstraction levels and terminology used should be tailored to suit the organization (Larman & Vodde 2010, Gorschek et al. 2007a, Anderson 2010). The meta-model of Gorschek et al. (2006) and the notion of "requirements work-up" – that is, creating higher level requirements as parents for orphan low-level requirements – are illustrated in Figure 2.9 below.

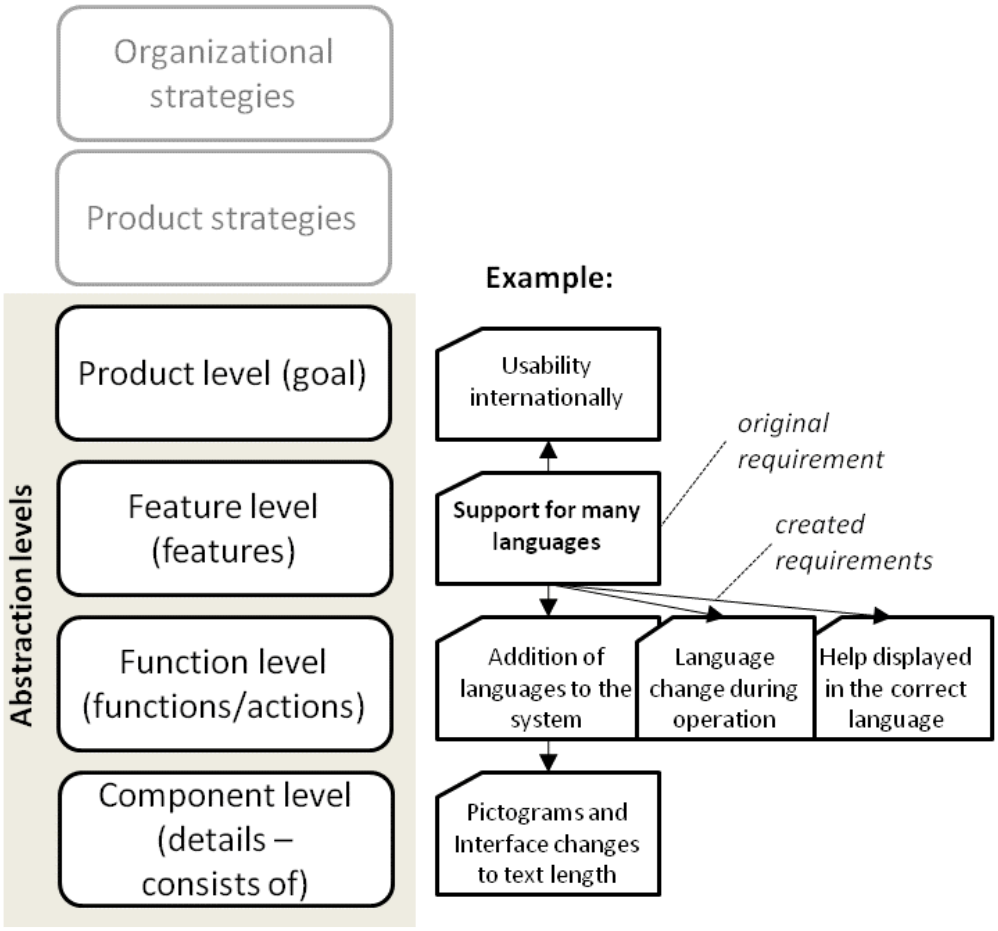


***Figure 2.9 Work item meta-model consisting of four abstraction levels and an example of "requirements work-up"; adapted from* (Gorschek et al. 2007a)**

According to the Gorsheck et al. (2006), the requirements abstraction model has been developed as a direct response to industry need which, as reflected on the agile practitioner-authors' recommendations discussed above seems quite plausible.

While there is little empirical evidence from the use of such meta-models in agile software development (Lehto & Rautiainen 2009), Gorshack et al. have evaluated the usefulness of their meta-model both in a controlled (Gorschek et al. 2007b) as well as an industrial (Gorschek et al. 2007a) setting with encouraging results.

According to (Gorschek & Wohlin 2006), using a meta-model for work item abstraction should help product management to better understand what should be developed and why, as the abstraction model helps organize the large number of requirements and requirement-like wishes that appear from various sources in varying levels of abstraction and detail. This helps planning and decision-making to adopt a proactive, strategic perspective that combines both short- and long-term views, instead of resorting to a reactive, project-based view (Gorschek & Wohlin 2006). Even in the case of missing or vague product strategies, using the requirement abstraction model has helped not only in linking the strategy to the requirements but in creating the strategy where it was lacking as

well (Gorschek et al. 2007a).

The work item meta-models for agile software development discovered in this literature review have been collected in Table 2 below. The models are organized to a synthesis of Cohn's planning onion (2005) and Smits' (2007) model of planning levels discussed in section 2.2.7 (*Levels of planning in agile software development*, page 26). Although not explicitly designed for agile software development[22], we have included the meta-model by Gorschek et al. (Gorschek & Wohlin 2006) as well for the sake of comparison.

Table 2 Work item meta-models for agile software development

		Meta-model					
		(Leffingwell 2011)	((Shalloway, Beaver & Trott 2009)	(Galen 2009)	(Keith 2010)	(Vlaanderen et al. 2011)	(Gorschek & Wohlin 2006)
Planning level (Smits 2007, Cohn 2005)	*Company strategy*	Level of investment in a strategic product theme	Vision	Vision & mission	(Vision)	Vision	-
	Product portfolio						-
	Product roadmap	Epic	Business capability	Epic	Saga Epic	Theme	Goal
	Release	Feature	Feature			Concept	Feature
	Iteration	Story	Story	Story	Story	Requirement definition	Function
	Day	Task	Task	Task	Task	-	Component

Except for Gorschek and Wohlin's model (2006), the meta-models in Table 2 originate from sources that discuss how agile development can be scaled beyond its original context of an individual, relatively independent team to support achieving enterprise-wide agility. However, the higher levels can also be used to express long-term business, product and release plans. For example, using the meta-model by Leffingwell (2011), a company's strategy would be reflected in the investment levels for its defined "strategic product themes". The roadmap for advancing a particular theme would be expressed in terms of epics and features[23], the planned contents of the current release would be fleshed out on the feature level, and the ongoing sprint in terms of stories and tasks. Note, that despite of the terms used in different meta-models, the work items on different levels can, and even should be thought of as user stories of different granularity (Leffingwell 2011, Pichler 2010).

2.2.10 Parent-child work item traceability in agile software development

The discussion so far seems to raise *planning levels* (section 2.2.7), *progressive refinement of work*

[22] At least parts of the case organizations may be striving for agile software development (Petersen & Wohlin 2010). However, this is not explicitly discussed by Gorschek et al.

[23] Leffingwell's (2011) model defines *vision* as simply a collection of epics.

items (section 2.2.8) and *using a suitable work item meta-model* (section 2.2.9) as cornerstones of planning in agile software development. However, the same cornerstones may also apply to progress monitoring on and upwards from the release level as well. For example, while a customer-facing product manager (Leffingwell 2011) may hardly understand the small user stories and technical tasks that make up the iteration backlog, the work item meta-model would help him relate the progress made to the marketable feature set(s) planned for the release. As another example, while the task level progress of individual sprints may not be of interest in a Scrum-of-Scrums meeting, reflecting the progress of the individual teams' stories to the higher level goal(s) that are common to the entire set of ongoing sprints certainly is[24].

From this perspective it is curious that the vast majority of the reviewed literature on agile software development, whether trade books or experience reports, *does not seem to take a clear stand on whether – or when – it is important to retain the trace of how the smaller work items were split from the larger ones*. In the few sources that do touch the issue, a considerable amount of ambiguity surrounds the topic. This is further explained in the paragraphs that follow.

Larman and Vodde (2010) recommend using meta-models to explicate item sizes, and identify *cell-like* and *treelike* splitting as two types of work item splitting. In cell-like splitting, the parent item (referred to as *ancestor item* by the authors) disappears. In treelike splitting, a record of parent-child relationships is preserved. According to the authors, the simplicity of cell-like splitting can be, depending on the context, both an advantage as well as a disadvantage. In both modes of splitting, independent prioritization of child items is equally possible, product owners may be influenced by the explicit relationships retained in treelike splitting making them subject to (incorrectly) view that the family of items must be prioritized as a whole, or that all child items must be done in order for the parent item to be done, thus reducing flexibility. When retaining parent-child relationships is important, at most one meaningful (that is, direct or indirect along the path back to the ultimate root item) parent should, according to the authors, be maintained. Furthermore, maintaining more than three levels of item splitting is recommended to be avoided. According to the authors' experience, having many nested levels of split items not only results in extra work in maintaining the relationships, but also but also tends to lead into the trap of technical, task-like work items instead of proper customer-centric, goal-oriented work items. However, why or how this happens is not explained. Larman and Vodde conclude that in general, cell-like splitting should be preferred over treelike splitting. This is in contrast with the view taken in requirements engineering research (e.g. (Berander & Jönsson 2006))(Berander & Jönsson 2006)

.

According to Keith (2010) "identifying a hierarchy of epics is valuable in communicating the big picture". Stober and Hansmann (2010) dedicate two pages (pp.77-78) for discussing the importance of having a system of hierarchical goals from the "overarching company strategy down to the actual coding work" but do not relate this to backlog management. Anderson, while discussing the use of a physical wall and cards for work item tracking, recognizes the need for large lean/agile projects to use hierarchical work item meta-models "as many as three levels deep" in order to support keeping the flow of value smooth (Anderson 2010). According to Anderson, it is typical that only the work items on the highest two levels are tracked on the Kanban board, as tracking the lowest level – development tasks – is less intresting from the perspective of tracking value stream and performance (Anderson 2010).

The prevailing dichotomy of whether or not to retain parent-child item relationships is important is

[24] For more on this topic, refer back to section 2.2.5 (*Out-of-the-box agile relates poorly to product management*) on page 26.

particularly well exemplified by the following excerpt (emphasis added):

> ***After an epic is split into smaller stories, I recommend that you get rid of the epic.*** *Delete it from the tool you're using or rip up the index card. You may choose to* ***retain the epic to provide traceability****, if that is needed. Or* ***you may choose to retain the epic because it can provide context for the smaller stories created from it.*** *In many cases, the* ***context of the smaller user stories is obvious because the epics should be split in a just-in-time manner*** *[…]. When an epic is ripped up and turned into smaller user stories shortly before the team begins work on it, remembering the context of the small stories is much easier.* (Cohn 2009), p. 178

As we see, Cohn does not come to a conclusion whether – and in what kinds of situations – retaining the trace is of significance. Also, while Cohn acknowledges that "some epics are so large that they split into epics", the implications of this are not further discussed.

So, the question remains: does retaining parent-child work item traceability support or hamper requirements engineering as purported by agile methods? To better understand this, we examine whether answers could be found from research on goal-oriented requirements engineering in section 2.2.11 below.

2.2.11 Parent-child work item traceability in goal-oriented RE

Goal-oriented requirements engineering (GORE or goal-oriented RE from hereon) is concerned with the use of goals for eliciting, elaborating, structuring, specifying, analyzing, negotiating, documenting, and modifying requirements (Lamsweerde 2001).

Similar to the work item meta-models discussed in section 2.2.9, goals may be formulated at different levels of abstraction, ranging from high-level, strategic concerns to low-level, technical concerns (Lamsweerde 2001). Goal-oriented requirements elicitation is an activity that continues as development proceeds, as high level goals (such as business goals) are refined into lower level goals. Eliciting goals focuses the requirements engineer on the problem domain and the needs of the stakeholders, rather than on possible solutions to those problems (Nuseibeh & Easterbrook 2000), and treelike splitting of goals is considered good for answering questions about how goals were achieved and why they were attempted (Ramesh & Jarke 2001).

The use of goals in GORE aims in driving requirements elaboration, achieving traceability from strategy to requirements, providing a criterion for sufficient completeness, avoiding irrelevant requirements, explaining requirements to stakeholders, choosing among plausible design alternatives, managing conflicts among multiple viewpoints and separating stabile from more volatile information (Lamsweerde 2001). For example, a requirement represents one particular way of achieving some specific goal; the requirement is therefore more likely to evolve than the goal itself; likewise, the higher level a goal is, the more stable it is likely to be (Lamsweerde 2001).

We see that all of the above mentioned qualities of GORE can in principles be thought to apply to progressive refinement of work items in agile software development as well. However, there are several crucial differences. While agile software development focuses on *user goals* (Larman & Vodde 2010), GORE focuses on *system goals* (Lamsweerde 2001). And, more importantly, most of the past work in GORE focuses on improving the requirements elicitation and elaboration phase in software development projects using a sequential (that is, a waterfall) life-cycle (Ktata & Levesque 2009, Lamsweerde 2004). Consequently, the practices recommended in GORE seem document-heavy, and as such, not directly applicable in agile software development.

More recently, however, GORE has been discussed in the context of software product development and iterative processes (Fricker, Gorschek & Glinz 2008) and large-scale agile software development (Ktata & Levesque 2009). Fricker et al. (2008) see GORE as helpful in linking

product management and agile software development:

> *Conceptualizing the interface between product management and development in terms of a goal-oriented system allows increasing efficiency and effectiveness of requirements communication. With feedforward, requirements can be specified with more impact and less effort. With feedback, the understanding of requirements can be assessed and managed. […] Feedback can be used by the product manager to support feedforward. Feedback allows him to learn about a team's expertise and resources to anticipate how to formulate requirements for that team. Such a combination of feedforward and feedback, called the full-information paradigm, corresponds to the most powerful paradigm for achieving high performance.*

Applying the terminology from the article by Fricker et al. (2008), the proper use of the product backlog, product vision, levels of planning, work item meta-models and treelike splitting as discussed in sections as described in sections 2.2.6-2.2.10 could be viewed as a "full information goal-oriented system".

Ktata and Lévesque (2009) see goal-oriented RE and using hierarchical systems of goals as a viable "avenue" for improving the business-development linkage in large-scale agile software development. They propose that stakeholders in large-scale agile projects should state their expectations in terms of goals, with *goal-oriented requirements language* (Lamsweerde 2004) used to reduce the level of ambiguity when needed. They also view that linking low user stories and features to tactical and strategic goals (that is, "the business perspective") as a necessity for large projects. According to Ktata and Lévesque (2009), this would require:

> *a hierarchical product backlog, in contrast with the current linear product backlog* [would] *bring better visibility, address both the technical as well as the business perspectives and provide a way to share knowledge across teams in a more agile and maintainable way than software documentation can do due to the inherent technical details.*

Ktata and Lévesque (2009) view that using traceable, hierarchical work item structures, development can better relate the progress made in terms of working software to how the higher level goals are being satisfied, and thus, potential business value. Based on this, goals can be prioritized on each level (Berander & Jönsson 2006), allowing the stakeholders to focus on prioritizing business goals rather than thinking about possible technical solutions, and when enough value has been achieved in terms of satisfying the most important goal, development can move to satisfying the next one. Ktata and Lévesque (2009) also state that "prioritizing same level items is far easier than prioritizing the whole product backlog in linear form". This is in line with the discussion in section 2.2.9 (*Progressive refinement of work items and work item meta-models*) on page 29. The authors conclude that the overall measure of success in agile software development should be "redirected from working software to achieving business goals".

To conclude, it seems that in contrast to some (but not all) expert practitioner-authors' opinions presented in section 2.2.10, researchers tend – at least in the context of market-driven software development – to consider treelike splitting and retaining the parent-child work item trace as crucial to effective requirements prioritization.

2.2.12 Linking release planning, roadmapping and the product backlog

In this section we, based on the reviewed related work, provide one way of seeing how the core processes of software product management – release planning and roadmapping – can be understood in the context of agile software development.

As discussed section 2.2.10 (*Parent-child work item traceability in agile software development*, pp. 31-33), there are plausible reasons for avoiding hierarchical work item structures or at least keeping them simple (Larman & Vodde 2010, Anderson 2010). However, requirements engineering researchers seem, at least in the context of market-driven (though not necessarily agile!) software

development to favor using and maintaining hierarchical work item structures. But another, and perhaps a stronger point in favor of using and maintaining hierarchical work item structures in agile software development is that doing so could directly address the reported practical concerns related to linking product management and agile software development (see section 2.2.5 *Out-of-the-box agile relates poorly to product management*, pp. 18-21).

The author takes the position that in the light of the discussed research, experience reports, trade press books written by expert practitioners and the research conducted in our research group (Lehto & Rautiainen 2009, Rautiainen, von Schantz & Vähäniitty 2011) it seems safer to conclude that the trace of how the smaller, 'child' user stories have been split from the higher level, 'parent' user stories is important to retain. This trace also provides a conceptual link with the product management processes of release planning and roadmapping. This position is reflected in publication VI as well.

Without the trace of how the individual iteration-level work items contribute to higher level objectives, monitoring development progress in terms of these higher level objectives becomes very difficult. We suspect that the missing feedback loops from iteration-level work items to the release plan and the product roadmap and back are, in fact, the missing link between agile software development and product management. Furthermore, explicitly linking the higher level objectives and the business context to iteration level work items may be useful for providing guidance for the developers' decision-making regarding the implementation details as well (Lago, Muccini & Vliet 2009).

In the light of the discussed related work, this dissertation defines *release planning*, *roadmapping* and the *roadmap* in the context of agile software development as follows:

> **Release planning** refers to the planning and continuous refining of the contents of the immediately upcoming release.
>
> **Roadmapping** refers to grooming the product backlog so that the high level goals for the foreseeable future are visible and reflect the current understanding of the relevant stakeholders.
>
> **Roadmap** is a view into the product backlog that depicts how a (line of) solution(s) is currently planned to evolve in terms of high level goals.

Figure 2.10 visualizes the essential properties of the product backlog and work items as discussed in sections 2.2.6-2.2.11 and in the light of the above definitions.

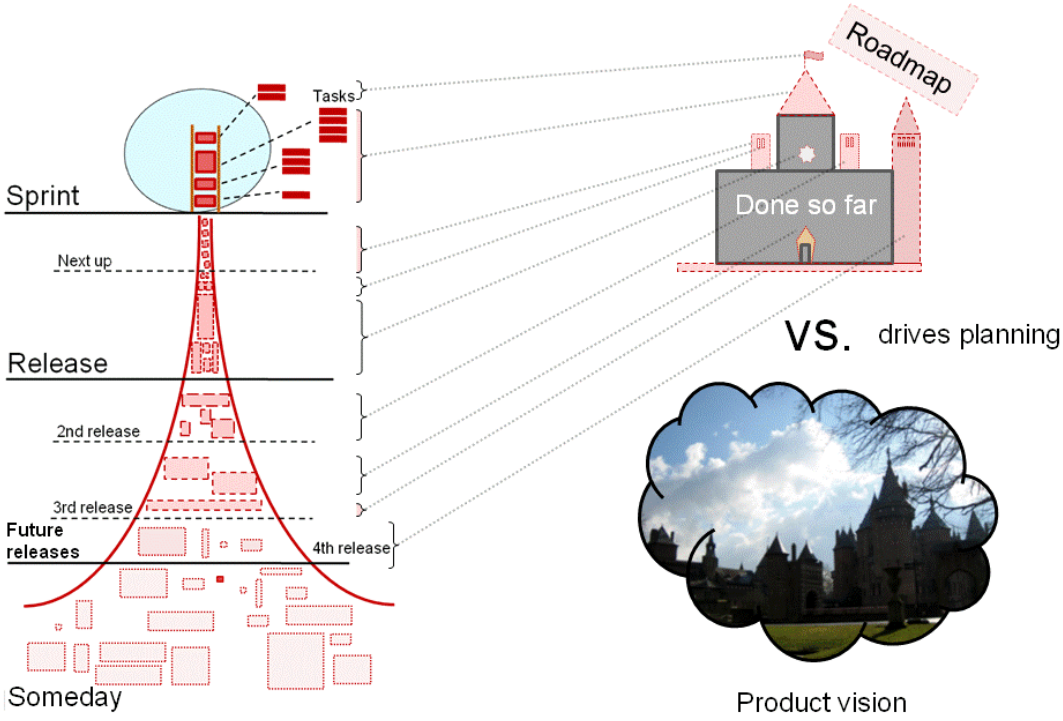


Figure 2.10 The product roadmap highlights the high level goals from the product backlog

As displayed in Figure 2.10, the product backlog consists of sections: *sprint*, *next up*, *release*, *future releases* and *someday*. Planning is driven by comparing an up-to-date vision to what has been accomplished so far[25]. The progress and/or planned timing of roadmap level work items can be viewed directly from the product backlog, as the trace between the business goals, roadmap level items and the smaller work items and tasks is kept explicit. Via this trace, the planned contents of future releases can also be seen, either in terms of roadmap-level work items or smaller, more detailed work items – if such have already been planned. Thus, while the product backlog is still a prioritized list, it should also contain the necessary information so that its contents can also be viewed as a hierarchical, tree-like structure.

The issues of traceability and hierarchical work item structures are further discussed in the results chapter in section 4.5 (*Linking long-term planning and daily work*, pp. 91-109).

In the following final subsection on linking product management and agile software development below, we examine what has been published regarding work item management tool support in agile software development and discuss the implications of the above definitions.

2.2.13 Implications for work item management tool support

In this section we discuss the material regarding tool support for work item management discovered in this review in the light of the of the conceptual integration of product management and agile software development as proposed in the previous section (*2.2.12 Linking release planning,*

[25] The grey parts of the castle ('Done so far') illustrate what has so far been accomplished.

roadmapping and the product backlog).

According to Mike Cohn (2009),

> *Much of how an organization works with its product backlog will be dictated by its* [work item management] *tool selection.*

The basic "tool" for work item and task management recommended both by many expert authors and as taught on commercial Scrum courses is writing user stories by pen on physical index cards, placing them on a wall, and then moving the cards typically from left to right as the development proceeds. This is in line with the original notion in eXtreme Programming by Kent Beck (Beck & Fowler 2001, Beck & Andres 2004). In practice, however, development organizations are often resorting to electronic tools that range from a simple spreadsheet to complex product management suites (Goth 2009). In the most recent source discovered in this review of the literature that discusses work item tracking using a physical wall and cards at length, Anderson (2010) notes that electronic work item tracking is from the team perspective optional, and the "more powerful" electronic tools allow for visualizing work item tracking as if a physical wall and cards were used. While Anderson sees physical work item tracking as more tangible and easier to implement on the team level, he continues (emphasis added):

> *For teams that are* ***distributed geographically or those who have policies that allow team members to work from their homes*** *one or more days per week,* ***electronic tracking is essential****.* … ***Electronic tracking is*** [also] ***necessary for teams that aspire to higher levels of organizational maturity****. If you anticipate the need for quantitative management, organizational process performance (comparing the performance across Kanban systems, teams or projects), and/or causal analysis and resolution (root-cause analysis based on statistically sound data),* ***you will want to use an electronic tool from the beginning****.* (Anderson 2010) pp. 72-73

In other words, there may be legitimate reasons for electronic tracking as well. Curiously, while Anderson does later in the book discuss hierarchical work items for scaling visual tracking to larger projects or across a project portfolio (pp.150-155), he does not relate the advice to using electronic tools. Based on the literature discovered in this review it remains unclear whether physical work item tracking would actually suffice even for a small organization striving to adopt agile/lean across its project portfolio.

However, as work items are split and interdependencies are found, keeping track of work items and progress by hand becomes more and more difficult (Schiel 2009). When it comes to electronic tools, Larman and Vodde (2008) recommend that agile development organizations should avoid traditional requirements management database tools. According to the authors, these "make it hard to have [the product backlog as] one force-ranked list". Larman and Vodde go on to argue that even most of the "agile tools" for work item management are "optimized for reporting rather than for real value work" because it is senior management who makes the decision whether to purchase a tool (Larman & Vodde 2008). They also recall that developers and product owners often "find [work item management tools] awkward and report [the tool] slows them down". Moreover, the Larman and Vodde (2008) point out that electronic tools "inhibit the lean Go-see practice".

If electronic tools are to be used, Larman and Vodde (2008) advocate the use of spreadsheets even in large-scale development:

> *Not because a spreadsheet is so good; just because it is better than the alternatives.* (p. 225)

Larman and Vodde (2008) emphasize their point by reporting that spreadsheets were adequate for running a multi-site product development in three countries with 400 people, and state that when it comes to backlog management, "the tool is seldom the real problem".

Woodward (2010) agrees with Larman and Vodde (2008) by pointing out that "filling out fields in a tool is not Scrum" and that "tooling should be kept as simple as possible". However, instead of spreadsheets, Woodward recommends using a dedicated tool for work item management (Woodward, Surdek & Ganis 2010). Pries and Quigley (2010) argue that spreadsheets "do not provide adequate support for backlog management" and view that using a database is necessary.

To contrast Larman & Vodde's (2008) success story of work item tracking in a 400-person project using spreadsheets, Vanhanen et al. (2003) report that even in a small company of less than 20 people, spreadsheet-based solutions for work item management using spreadsheets became awkward: moving items to and from the sprint backlog, updating estimates, making reports, logging and tracking effort spent and preserving the backlog history were considered difficult. While it could be argued whether some of these needs (for example, tracking spent effort) are essential for agile software development, Vanhanen et al. report also that tracking the sprints' and releases' goals was challenging as well (Vanhanen, Itkonen & Sulonen 2003). In retrospect, this was at least partly because the spreadsheet-based solution in question did not support progressive refinement of work items. In line with this observation, Gorschek et al. (2007a) note that in one of their case companies the use of spreadsheets for requirements management initially prevented the full-scale adoption of the requirements abstraction model (see section 2.2.9), and adopting a new tool was necessary. In the other case company where a "traditional requirements management tool" capable of hierarchical requirement structures was in use, this particular problem did not occur (Gorschek et al. 2007a).

According to Lehto and Rautiainen (2009), in a medium-sized company striving for enterprise-wide adoption of agile software development, people on different organizational levels (for example, in product management and in development teams) used separate spreadsheets with different work items for managing the backlog of a single product, making the tracking of development progress to against business goals very difficult.

Dedicated work item management tools are "becoming a common fixture in agile development environments" and especially in larger projects having multiple teams (Schiel 2009). And, despite the emphasis of the agile manifesto of *individuals and interactions over processes and tools*, the market for electronic tools for work item management in agile development "is booming" (Goth 2009). Thus, whether this is good or bad for practicing and/or adopting agile software development, it is reasonable to assume that at least those organizations that are already using spreadsheets for work item management are likely to adopt dedicated work item management tools sooner or later. This means that the ability to select a tool that supports proper backlog management as described in the literature (and summarized in sections 2.2.5-2.2.12) may be crucial. Thus, we conclude this section by discussing guidelines for selecting a work item management tool in the light of both existing literature, as well as our definitions of roadmapping and release planning from section 2.2.12.

Cohn (2009) provides two guidelines that apply to work item management "regardless of which tool is used". First, there should be a single product backlog per product. Second, the product backlog should be kept to a reasonable size (at most 100-150 work items) by the means of progressive refinement of work items and grouping of similar items into themes[26]. In addition to the above suggestions by Cohn, this review discovered two books written by expert practitioners (Woodward, Surdek & Ganis 2010, Schiel 2009) that explicitly stated criteria for selecting work

[26] Cohn sees the product backlog as a flat list, with a "theme" being a label that can be used to group work items together. Like Cohn, most expert practitioner-authors do not discuss the possibility of *having different views* (e.g. one showing a flat list and the other the item hierarchy) into the product backlog.

item support tools.

The following list of important features for electronic work item support tool has been combined from these sources:

1) The tool should support teams to build and maintain their sprint backlogs.
    a. Creation, ownership, estimation, and changing the status and remaining hours of tasks (Woodward, Surdek & Ganis 2010, Schiel 2009)
    b. An electronic task board visualization (Schiel 2009)
2) The tool should be accessible off-site by all project personnel (Schiel 2009)
3) Automatic generation of sprint burndowns and supporting project status reporting using a combination of product and sprint backlog data (Schiel 2009)
4) Supporting product owners to create, display, revise and delete work items (Woodward, Surdek & Ganis 2010, Schiel 2009)
5) Multiple views into the product backlog; for example, "a themed (or otherwise tagged) view to support a variety of different viewing needs" (Schiel 2009)
6) Allow for disaggregating work items into smaller work items (Woodward, Surdek & Ganis 2010)
7) Allow for easy prioritization of work items (Woodward, Surdek & Ganis 2010, Schiel 2009)
8) Allow product owners to allocate product backlog items into releases (Schiel 2009)
9) Allow for estimation of risk, value and cost with flexible units (Schiel 2009)
10) Alerting proper personnel when work items are not properly prepared for sprint planning (Schiel 2009)
11) Allow for recording and maintaining work item dependencies and alerting proper personnel when items are in danger of being out of sequence (Schiel 2009)
12) Allow for recording the stakeholder(s) of a work item (Schiel 2009)

From the perspective of linking roadmapping and release planning with the product backlog as proposed in section 2.2.12, properties *5)* and *6)* seem the most interesting. Schiel (2009) does not elaborate on what is meant in property 5) by "a themed view to support a variety of different viewing needs", but from the context can we speculate that one of these needs might be the ability to highlight the roadmap from the multitude of work items in the backlog. However, Schiel does not explicate whether hierarchical work items are necessary. Also, while Woodward in property *6)* mentions "disaggregation of work items into smaller work items", she does not explicitly tie this to progressive refinement of work items or hierarchical work items.

Based on this review of the literature, it would seem that the progressive refinement of work items is one of the most intricate issues in managing agile software development. Considering the number of subtleties related to the use of the product backlog (levels of planning, progressive refinement of work items, item meta-models and parent-child traceability; see sections 2.2.5-2.2.12), it seems quite plausible that – as stated by Larman and Vodde (2008) – the tool indeed is seldom "real problem". However, in this dissertation we take the following stand: once the organization (or the critical part of it) has grasped how the product backlog and the product vision are supposed to drive development, *proper tool support may play a key role in making of breaking the transition*.

We will return to discuss product management and work item management tool support in section 4.5 (*Linking long-term planning and daily work*) of chapter 4 (*Results*).

2.3 Portfolio management

To perform both product development and servicing, especially small companies must be able to "juggle" their scarce resources so that at each moment those activities that from a business perspective are the most important get attended to (Miettinen, Mazhelis & Luoma 2010, Nambisan

2001, Tikkanen, Kujala & Artto 2007). For example, the need to develop new functionality or provide training may suddenly appear if it is required to close a deal, realize channel partner revenue, or a certain customer simply wishes to purchase it (Hodgkins & Hohmann 2007).

Managing this kind of "juggling" is traditionally referred to as *new product development portfolio management* (Cooper, Edgett & Kleinschmidt 2002) or simply, portfolio management. Portfolio management has been extensively written about in literature on managing new product development (NPD), and potential for cross-domain knowledge sharing with software engineering is said to exist (Nambisan & Wilemon 2000). But does portfolio management integrate with agile software development, and if so, how? How should portfolio management be conducted in a context where some, but not necessarily all of an organization's activities are managed using an agile life cycle? And is a degree of portfolio management relevant for even small organizations and/or teams?

In the sections that follow, we examine what literature on software product management (section 2.3.1), NPD portfolio and pipeline management (section 2.3.2), as well as the emerging work on portfolio management in agile software development as represented by practitioner literature and experience reports published at conferences (sections 2.3.4-2.3.11) have to offer with respect to the above questions.

2.3.1 Portfolio management in software product management

Overall, literature on software product management sees portfolio management as a very high-level activity dealing with coarse-grained allocation of capital (Kittlaus & Clough 2009). In literature on software product management, portfolio management means making decisions about the set of new and existing products based on the market trends and the product development strategy, and establishing partnerships and contracts (Weerd et al. 2006b).

As an example, the software product management competence model (Bekkers et al. 2010) defines portfolio management as follows:

> *Portfolio management concerns the strategic information gathering and decision making across the entire product portfolio. Its first focus area is Market analysis, which gathers decision support information about the market needed to make decisions about the product portfolio of an organization. Secondly, Product lifecycle management concerns the information gathering and key decision making about product life and major product changes across the entire product portfolio. Finally, Partnering & contracting focuses on establishing partnerships, pricing, and distribution aspects in which the product manager is involved.*

The current body of knowledge on software product management seems to view portfolio management as *product portfolio management* and mostly ignores agile software development[27]. Thus, it is only natural that from the literature on software product management, virtually no advice was found on how new product development portfolio management should be conducted in conjunction with agile software development – be the organization large or small.

2.3.2 NPD portfolio management and small software companies

The concept of portfolio management originates from the context of large organizations, where activities are primarily organized as projects, there is an explicit strategy, personnel dedicated to managing the portfolio exist, and getting the products to the market involves manufacturing (Martinsuo 2001).

[27] For a detailed discussion, see section 2.2.4 (*Product management literature relates poorly to agile*, pages 24-26)

The relevance of portfolio management in the context of small organizations has not, however, been discussed (Martinsuo 2001, Lawson, Longhurst & Ivey 2006). It is plausible that in small organizations, resource allocation would be managed by a small group in a timely response to client demands, with the decisions being shared, communicated and understood without the aid of more formal approaches or tools (Lawson, Longhurst & Ivey 2006).

On the other hand, in small organizations the same people are responsible for managing a number of "portfolios" in addition to the portfolio of development projects. These other portfolios involve ongoing sales, deliveries and other services as well as relationships with customers and partners (Alajoutsijärvi, Mannermaa & Tikkanen 2000, Tikkanen, Kujala & Artto 2007), making the situation complex despite the fact that the total number of products and services offered may not match that of larger companies. And, while in small companies there is typically only a single or at most few products to be developed, there is a strong focus on servicing, support and maintenance (Wangenheim et al. 2006). This leads to having a *portfolio of activities* that require attention from the development people, and the basic problem may be quite similar to that of larger companies.

This review of the literature discovered no suitable advice for portfolio management in small software companies. However, what is known is that at some point, a small software organization will, in order to grow, need a management system with "layers of control, structures, and routines" (Miettinen, Mazhelis & Luoma 2010, Koivisto & Rönkkö 2010). And while many of the formal and structured approaches used in larger firms and prescribed in the literature may be inappropriate in the case of the small teams or organizations, lack of appropriate control and monitoring has been reported as "likely to lead to high levels of risk and failure" (Hoffman et al. 1998).

Thus, the question remains: is explicit portfolio management of development efforts relevant for small organizations and/or teams? This is the topic of publications IV and V, and we will return to it in the in section 4.3 (*Do small software organizations need portfolio management?*) of the Results chapter on page 86.

2.3.3 NPD portfolio management and agile software development

Thus, instead of abandoning traditional portfolio management, it models should somehow be combined with agile software development (Dybå & Dingsøyr 2008).

Traditionally, portfolio management is realised by integrating techniques for project evaluation, selection and prioritisation in a review process for ongoing development projects (Cooper, Edgett & Kleinschmidt 2002). Such review processes often organise product development projects into a sequential set of phases (for example, beginning with idea generation and ending with product launch and maintenance), with a corresponding business and prioritisation decision point (that is, the review) at the end of each phase. Development work is conducted during the phases, along with gathering the information needed to pass the next review (Karlström & Runeson 2006).

NPD literature presents two basic alternatives for implementing the portfolio management process (Cooper, Edgett & Kleinschmidt 2002). The first one – called *gates dominate* – emphasises decision-making through in-depth reviews for each ongoing development effort (or idea), and manages the portfolio in a bottom-up manner. The second approach – *portfolio reviews dominate* – is top-down, with decisions based on looking at the portfolio as a whole. The *gates dominate* model is described as better suited for larger firms in mature businesses with dedicated resources and fairly static portfolios, because the emphasis is more in making sound go/kill decisions for individual projects than re-prioritising the entire portfolio every few months. Likewise, the *portfolio reviews dominate* model is described more appropriate for fast-paced and fluid markets and smaller companies because it allows for more dynamic resource allocation through periodically reviewing

the entire portfolio (Cooper, Edgett & Kleinschmidt 2002).

Proper portfolio management has been recognized as challenging to implement "even for the best of organizations" (O'Connor 2004), and in general, literature on NPD portfolio management seems – based on this review of the literature – so far to be largely disconnected from agile software development. However, recent work on agile software development seems to be taking up the challenge. Several research papers, experience reports and books written by expert practitioners and consultants on how to set up portfolio management so that the result would compatible with agile and/or lean principles were discovered, and are examined in the sections that follow.

First, we examine how agile methods are reported to fit together with the *gates dominate* approach of portfolio management (section 2.3.4). Then, we based on recent practitioner literature and experience reports on agile software development identify how portfolio management can be viewed as occurring on many levels in an enterprise-wide adoption of lean/agile principles (section 2.3.5) and address each of these levels in detail (sections 2.3.6-2.3.10). We conclude the discussion of portfolio management and agile software development with reviewing proposed models for agile portfolio management that essentially seem to adhere to the *portfolio reviews dominate* approach (section 2.3.11).

2.3.4 Agile vs. "gates dominate" product development models

Literature on portfolio management is more concerned with the prioritization of products and projects instead of features (Larman & Vodde 2008). Also, development is often as a separate "phase" in the idea-to-solution cycle (Krebs 2008). Thus, many advocates of agile and lean product development tend to see much of the literature on new product development portfolio management as incompatible with agile software development. Or as put forward by Reinertsen (2009) in his recent practitioner book on lean product development:

> *Officially, many* [product development organizations] *have phase-gate processes. Work is divided into mutually exclusive phases separated by gates. One phase must be complete before the next one can begin. For example, such processes typically require that all product requirements* [can] *be defined before beginning design activities. The team appears to do this, and delivers complete product requirements to management at the gate review. Then, they are given approval to exit the requirement definition phase and to begin the product design phase. On its surface, this procedure appears quite sensible, and it seems to work. What really happens is quite different. When I survey managers in my product development courses, 95 percent will admit that they begin design before they know all requirements. In fact,* […] *product developers begin design when 50 percent of requirements are known. They simply do not publicize this to management. Instead, they engage in a time-honored ritual of asking for permission to proceed. There is a tacit "don't ask, don't tell" policy. Managers politely avoid asking if the next phase's activities have already begun, and developers discreetly avoid mentioning that they have already moved on. In practice, sensible behavior prevails, despite the presence of a dysfunctional formal procedure.* (p. 2)

However, while proponents of gate models agree that gate models have the danger of ending up as bulky and bureaucratic in practice, they also view that gate models are "just a guide that suggests best practices, recommended activities and likely deliverables" (Cooper 2009). Moreover, Cooper (2009) continues:

> *Process innovations made by leading companies since the introduction of the original Stage-Gate model in mid-1980s make gate models apt for today's rapid pace of product innovation as well.*

Modern versions of phased gate models (often referred to as *Next-Generation Stage-Gate*) are "integrated with the innovation front end and roadmapping, employ leaner gates, combine portfolio reviews with portfolio management techniques such as strategic buckets and scorecards, value stream mapping, have multiple tracks according to project type […] and allow for agile and spiral

development processes" (Cooper 2009).

To this, Larman and Vodde (2008) argue:

> *While Cooper's Stage-Gate of today is not the Stage-Gate of yesterday* […] *many companies 'installed' their stage-gate process years ago and did not bother keeping up with Cooper's evolution.* […]

In the experience of Larman and Vodde, most gate model implementations in large organizations exemplify practices that Cooper would refer to as "misconceptions about the Stage-Gate" (Larman & Vodde 2008, Cooper 2008).

Proponents, opinions, and speculation about the degree of industrial adoption of Next-Generation gate models aside, there is yet little evidence of agile methods co-existing (or not) with "gates dominate" type approaches to portfolio management. We discovered three research articles by Karlström and Runeson (2006, 2005) and Wallin et al. (2002) and three experience reports (Vidgen & Wang 2009, Laanti 2008, Tengshe & Noble 2007) in which agile software development had for a time co-existed with "gates dominate" type portfolio management. These are discussed below.

All three of the research articles feature the same case companies, Ericsson and ABB, with Kalström and Runeson (2006) offering the most detail regarding the case projects and the actual integration of agile software development and with the gate model. The authors conclude as follows (emphasis added):

> *It is indeed possible to successfully integrate project management models of the stage-gate type with XP projects. This has been achieved through complete isolation of the engineering teams and attempting to make these look as similar to a regular team as possible.* (p. 222)

As isolation is hardly the way to leverage the proposed benefits of agile methods for the entire organization (Kettunen 2007), the actual "success" of the integration seems unclear. The article offers additional detail that sheds more light on the matter. Regardless of whether agile software development was used or not, the gate models were not in practice adhered to – whether the projects were using eXtreme Programming or not. XP projects attempted to work with the gate models and fit their outputs accordingly, whereas development projects that employed a "traditional" life cycle model simply ignored the gate models when needed.

Key factors that were reported to hinder the integration of agile software development with gate models were as follows: management understanding of agile software development, missing a clear customer interface, unclear artefact incompatibility in terms of XP projects' outputs and what the gate model required, and the differences in the level of abstraction on which the development teams and the management communicated – or even wished to communicate (Karlström & Runeson 2006). As an example of the latter, management found the "good micro planning" produced by employing XP as something the development was supposed to resolve by themselves anyhow, since the management was concerned with high level product and business plans. And, interestingly enough, despite the relative success of both of the studied XP projects, the teams involved dispersed afterwards (Karlström & Runeson 2006). All of these observations can be interpreted as signs of incompatibility with "gates dominate" type portfolio management and agile software development.

The three experience reports (Vidgen & Wang 2009, Laanti 2008, Tengshe & Noble 2007) convey similar insights. Tengshe and Noble (2007) report an example of an agile team fitting its outputs to the deliverables required by a gate model, while Vidgen and Wang (2009) describe a team that has "adopted some agile-like practices to circumvent the restrictions of the waterfall processes imposed by the company" (pages 360 and 367). According to Laanti (2008) the synchronization of Scrum teams' output to the phased review process the entire program was supposed to adhere to was considered problematic.

Based on the discovered research articles and books by expert practitioners, we take the position that at least "gates dominate" type implementations of portfolio management and agile software development may have a hard time co-existing. Also, looking at the hurdles involved, they seem unlikely to disperse before the nature of agile software development – and how it should link with product and portfolio management – is understood throughout the organization.

2.3.5 Levels of portfolio management

While the focus of agile software development has been on the single-team-single-customer scenario, development cannot exist in isolation but must be integrated with the enterprise level decision-making processes such as portfolio management if the benefits of agile software development are to be leveraged for the entire organization (Karlström & Runeson 2006, Steindl 2005). According to Steindl (2005), this can lead the business to being "more responsive, focused, variable, and resilient". Furthermore, portfolio management has been reported to have an "instrumental" role both in allowing the transformation to take place and as well as in supporting the transformation (Tengshe & Noble 2007).

But just like product management needs to be seen in a new light, how portfolio management is understood needs to change as well (Hodgkins & Hohmann 2007). And this change may not be small:

> *Some organizations have a Project* […or] *a Program Management Office (PMO).* [According to PMBOK]*, their activities "range from providing project management support functions to actually being responsible for the direct management of projects". By looking at this description, it ought to be obvious that the PMO is no longer needed* […] *when Scrum is adopted.* [This is because] *most project management responsibilities are moved into the teams. Scrum support comes from the team of hands-on agile coaches* […] *Responsibilities of releases, metrics and other schedule- or content-related issues move to the Product Owner Team.* (Larman & Vodde 2008) p. 249 (emphasis added)

According to Larman & Vodde (2008), "most agile literature recognizes that a traditional Project Management Office is not needed in an agile and lean development organization". However, if this is the case, most of the literature discovered in this review seems to avoid dealing with the issue – which may not be too helpful for organizations trying to adopt agile software development. Current literature seems to provide little advice as to how portfolio management should be understood in agile software development, or how to deal with a situation where some, but not all of the activities that require attention from the development people are run using (for example) Scrum.

Looking at the discovered literature that does discuss agile software development and portfolio management, we see that different authors seem to be using the term 'portfolio management' referring to different kinds of decision-making – and on different levels. There is not a single, but actually multiple different kinds of "portfolios" to manage. In top management, decisions are made regarding the investments into the current and possibly new products and services offered by the company, in other words, the product portfolio. In product management, decisions are made regarding the Epics and Features (see section 2.2.9) to be developed in the current and upcoming releases. In managing a release project, there is a portfolio of features that are planned to be included in the release, as well as other work that needs to be taken care of for a successful release. On the level of an individual team and development iteration, the portfolio of work is made up from the committed stories as well as possible other duties the team members may have. And an individual developer has to make decisions on what tasks (some of which may or may not be related to the release he is working on) he will take up and when. This notion of *levels of portfolio management* is illustrated in Figure 2.11 below and explained in more detail along with references in sections 2.3.6-2.3.10.

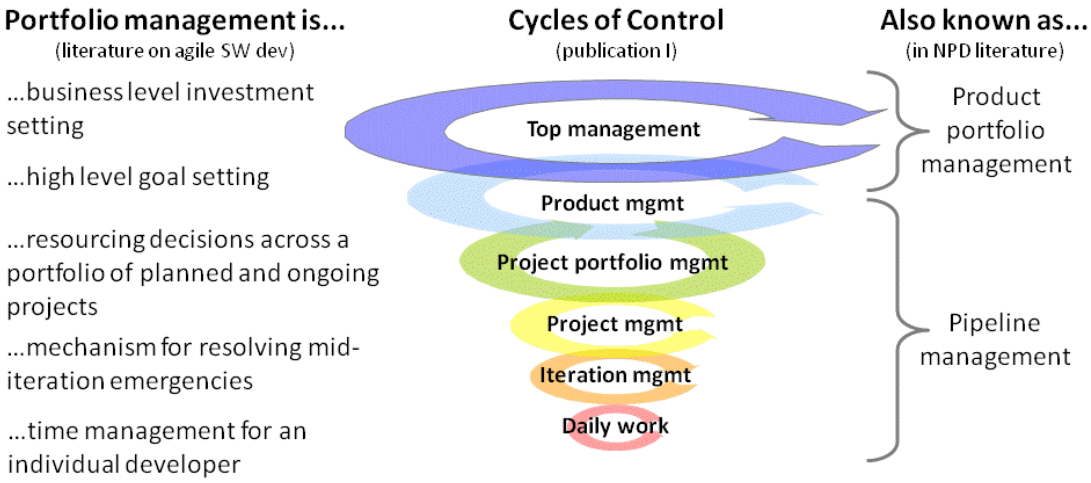


Figure 2.11 Levels of portfolio management found from literature on agile software development

Figure 2.11 also depicts how the levels of portfolio management correspond to product portfolio management and pipeline management as understood in the NPD literature (Kahn, Castellion & Griffin 2005).

We take the position that viewing portfolio decision-making through the lens of agile software development planning levels[28] (see section 2.2.7) offers a possible framework for analysing and discussing portfolio management in the context of agile software development. The following discussion in sections 2.3.6-2.3.10 below examines each of the portfolio management levels in more detail. After this, we in section 2.3.11 conclude the discussion on portfolio management and agile software development by reviewing the specific models for agile portfolio management we found from the literature.

2.3.6 Portfolio management as investment level setting

At the top, portfolio management in an organization striving for agile software development is about setting investment levels for business areas of the enterprise. For example, Dean Leffingwell's framework for agile requirements and enterprise agility (Leffingwell 2011) considers portfolio management as a top-level activity responsible for defining the enterprise's *strategic investment* (or *product*) *themes*[29] (or *investment themes*, or simply *themes* for short) and their *investment levels.* Themes drive the enterprise's investment in systems, products, applications and services. Thus, they are the highest level but still tangible manifestation of enterprise strategy – or business strategy, if there's only a single business unit (Leffingwell 2011). This view of portfolio management corresponds roughly with the view of portfolio management typically adopted in

[28] The planning levels in agile software development, as well as the levels of portfolio management, of course also correspond to the levels of the Cycles of Control framework (Rautiainen 2004) as described in publication I. Hence, we did not use Cohn's planning onion (2005) in Figure 2.11.

[29] As opposed to most other authors, for example (Galen 2009, Keith 2010, Cohn 2009), Leffingwell (2011) does not use the term 'theme' to refer to a group of related user stories. Also, while Leffigwell (2011) uses the term *strategic investment theme*, his earlier writings refer to the same concept as *strategic product theme*.

literature on software product management (see section 2.3.1).

Themes differ from work items in that they are not prioritized using rank order, but rather as percentage-based investment levels. While a work item that has a low priority may not be ever get acted on, the theme with the lowest relative investment level still should be addressed over time if the enterprise acts according to the longer term priorities it has decided on. Thus, for a focused organization, only a few themes should be active at any one time (Leffingwell 2011). Also, themes have a longer time span than work items, and may be largely unchanged for up to a year or even longer (Leffingwell 2011).

Figure 2.12 provides a fictitious example of what Google's strategic product themes might have looked like at a hypothetical business unit responsible for development of the web applications in question.

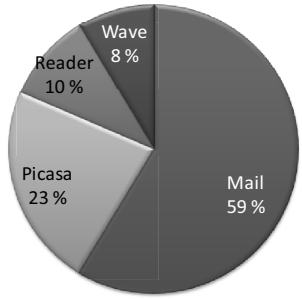


Figure 2.12 An example set of themes and their investment levels

2.3.7 Portfolio management as product and business goal prioritization

Portfolio management also takes place in the form setting, revising and prioritising high level goals for the business areas so that they from the enterprise's perspective form a coherent whole that is compatible with e.g. the current investment level. This means dealing with possible competing roadmaps (Kalliney 2009, Hodgkins & Hohmann 2007) and forming a "roadmap of roadmaps" as needed (Wilby 2009). Overall, this draws the boundaries for planning release projects (Lehto & Rautiainen 2009).

In Leffingwell's framework (2011), the prioritization of *Epics* occurs on this level. Epics are (typically) derived from themes, and are the highest level expression of a customer need (see Table 2 *Work item meta-models for agile software* development on page 31).

Epics are defined as stories which are estimated as "too big to be realized in a single release". The instantiation of themes occurs first through Epics, then Features and finally through Stories (see Figure 2.13).

Figure 2.13 Portfolio management at the business level decides on themes and epics (Leffingwell

2011)

Setting, refining and revising the criteria used for prioritising projects can also be considered to belong to this level (Azar, Smith & Cordes 2007). Table 3 below presents example criteria formulated to guide project (and feature) prioritization in a small development organization adapted from (Azar, Smith & Cordes 2007).

Table 3 Example criteria to guide project/feature prioritization (Azar, Smith & Cordes 2007)

Criterion	**Definition**
Sales value	Value to the sales organization in terms of the potential to generate new sales
Customer satisfaction	Value to existing customers; no direct revenue implications, but indirect implications in terms of existing customers buying more, renewing maintenance contracts and recommending the product to others
Marketing value	Drawing attention and recognition for marketing purposes
Strategic value	Reaching the organization's strategic objectives, such as becoming business partners with other companies through integrations
Integrity value	Integrity issues, such as data or business integrity

2.3.8 Portfolio management as project resource allocation

The most common interpretation of portfolio management in the literature on agile software development is to perceive it as responsible for the *resourcing decisions across a portfolio of planned and ongoing activities*, such as projects.

Shalloway's (2009) concept of *Lean portfolio management* means deciding on a relatively frequent basis on how the development resources are allocated across a portfolio of projects in order to develop and deliver those *minimum marketable features* that at the moment seem to provide the most business value. Thus, Shalloway's portfolio management refers to short-term tactical project-wise resource allocation, which differs considerably from e.g. Leffingwell's (2011) use of the term.

Figure 2.14 illustrates Shalloway's approach of how the development resources are allocated to develop the most important business features.

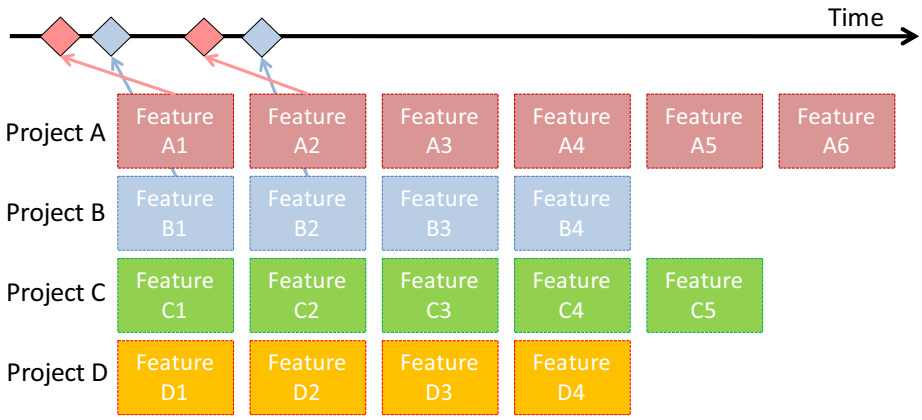


Figure 2.14 Portfolio management as decision-making on short-term project resource allocation (Shalloway, Beaver & Trott 2009)

Figure 2.14 displays four projects (A-D), listed starting from the most important project (at the top) All of the projects have prioritized backlogs (the leftmost box is the feature deemed most important). The projects may be about developing the next version of a particular product, or they can be just some other development work, for example, working on a systems integration effort for a particular customer. In the first iteration, the most important business features (A1 and B1) from the two most important projects (projects 1 and 2) are implemented, while the rest of the projects remain on hold. Since Shalloway's *business features* can be developed in an iteration, they are smaller than Leffingwell's *features*, which, by definition, do not fit in an iteration but do in a release (Leffingwell 2011). Thus, in terms of Leffingwell's framework, Shalloway's *business features* roughly correspond to a group of related stories small enough to fit in an iteration.

Pichler (2010) recommends that "competing backlogs" should be dealt with in a similar fashion as recommended above. Another voice in favour of this kind approach comes from Larman and Vodde (2008). They note that for organizations of less than 100 people, "prioritizing on the level of the portfolio of products and services offered leads to local optimization". Instead, portfolio management should be carried out by merging the backlogs for different product/service offerings into a single backlog and then performing backlog management as usual. Likewise, Krebs (2008) advocates that the ongoing and planned projects should be kept is kept in a list called the "project portfolio backlog". Decisions about which projects will continue, put on hold, launched or killed are then made on a per-sprint basis. A similar approach is also mentioned by Rothman (2007):

> *If you develop in iterations and always develop the highest priority requirements first, you can change* [project] *priorities as often as you finish an iteration. I'm not recommending that you do so but that you could.* (p. 319)

The approaches discussed above seem to assume that the iteration cadence for the entire enterprise is synchronized[30]. Also, they seem to assume all of the ongoing activities that require the developers' attention follow an agile life cycle and have up-to-date backlogs. The common mindset in the literature seems to be that when all of the development activities are run using an agile life cycle, the progress and potential value of each activity are transparent, and thus, portfolio

[30] Other expert practitioner-authors on agile software development also discuss the synchronization of ongoing activities' cadence to make portfolio management easier, for example, Keith (2010) and Cohn (2009). This is discussed in publication V as well.

management becomes easier. This is exemplified by the following excerpt from Rothman (2009):

> *If you are already using an agile approach for your projects or an iterative or incremental life cycle where you have an opportunity before the end of the project to finish features, you can use the ideas here to be a successful leader in the organization, no matter what level you are. If you use a serial life cycle where you can't see any progress until the end of a project, you will find these ideas more difficult to use. If you use a serial life cycle, try to create interim deliverables. The more frequently the projects deliver something you can see, the easier it will be to manage the project and to manage the project portfolio.* (p. 24)

While this is both plausible, as well as something to be strived for in practice according to recent experience reports (Laanti 2008), in most organizations, all, or even most of the development resources' activities are not run using an agile life cycle – or even conducted in distinct projects (Rothman 2009). However, we found little discussion of this in the literature beyond the occasional warning against such situations. For example:

> *Don't mix agile and non-agile projects in one portfolio.* (Krebs 2008), p. 137

According to Cohn (2009), "Opinions are split on whether Scrum and sequential development approaches can coexist forever". While organizations are doing this anyway, and at least in large organizations it is temporarily necessary, Cohn argues that the "fundamental differences" between agile and sequential cause conflicts to become "more and more painful". Cohn speculates on what will happen if the sources of these conflicts are not removed:

> *Organizational gravity will pull the organization back to whatever software development process was in place before adopting Scrum. A few nonthreatening agile practices such as daily scrums or continuous integration might remain, but the organization will have been unable to achieve the compelling benefits of becoming agile.* (Cohn 2009)

On the other hand, it is equally plausible – if not even more so – that different types of project management life cycles should be allowed to co-exist. According to Mark Price Perry (2009):

> *[…] since projects and their characteristics can be extremely different, project classes are critical to both enable the application of different methodologies, both traditional and Agile, and to provide the project teams with what they need to achieve. Those that suggest agile is really just winging it are mistaken. Likewise, those that contend that agile project management is suitable for any and all projects of the PMO are clearly misguided, though perhaps well-intentioned. It would be a huge mistake to categorically dismiss one approach over the other and it would foster undesired organizational behavior and project teams would find ways to work around the mandated approach.* (p. 134)

Thus, based on this literature review, the question whether agile and sequential can or should co-exist remains open. Based on the material discovered and reviewed in this literature review, and the fact that most companies are practicing "mixing agile and non-agile projects" anyway, it would seem wiser to at first strengthen the role of explicit portfolio management than to immediately judge one way or the other based on unclear evidence.

Furthermore, even if agile and non-agile projects remain "unmixed", it may be difficult for those in charge of the project portfolio – this could be a cross-functional team (Hodgkins & Hohmann 2007) or a team composed of product owners (Larman & Vodde 2008) – to prioritize the projects' backlogs against each other on a sprint-by-sprint basis. First of all, there is the pre-condition of having all backlogs up-to-date and ready to be assessed against each other. Second, if the higher level context for the small work items is not evident, it may be difficult to trace the trade-offs back to business goals[31]. Based on the experience reports – for example (Kalliney 2009) – as well as the

[31] For more on the issue of work item child-parent traceability, refer back to sections 2.2.10-2.2.13.

advice in recent practitioner books reviewed in section 2.2 (*Product management*), we suspect that real-world organizations may have a hard time meeting either of these pre-conditions.

Thus, this level of portfolio management may be the most challenging one to succeed in practice.

2.3.9 Portfolio management as resolving mid-iteration emergencies

Although literature on agile software development is generally against the notion of making mid-iteration changes to iterations' staffing or contents (Larman & Vodde 2010, Hodgkins & Hohmann 2007, Tengshe & Noble 2007, Larman & Vodde 2008), portfolio management is responsible for resolving mid-iteration emergencies that require escalation. Because of concerns regarding the customer satisfaction of important clients or revenue, it can in some situations benefit the organization to "raise the red flag" and adjust the resourcing for the remainder of the iteration to deal the crisis even if this compromises the completion of what was being worked on (Rothman 2007). In these situations, stripping ongoing activities of resources in and/or putting them on hold in order to salvage something of more importance requires portfolio management decision-making.

2.3.10 Portfolio management as time management in daily work

The reviewed literature on agile software development in general recommends that all of the team's work, whether related to the ongoing development effort or not, should be included in the sprint backlog, and that a single person should have a single sprint backlog to pull tasks from at any given time (Larman & Vodde 2010). However, when multiple assignments must be made, the literature seems to be strongly in favor of assigning stabile teams to pull work items from multiple backlogs, as opposed to assigning people on multiple teams (Tengshe & Noble 2007, Larman & Vodde 2008). And according to the reviewed experience reports, when it comes to multiple assignments, both modes of operation are commonly applied in practice (Hodgkins & Hohmann 2007, Laanti 2008, Tengshe & Noble 2007).

Thus, in the less-than-optimal but possibly all-too-common-situation of teams or people having multiple responsibilities, the bottom-up time management of the daily work of a team or even individual workers can be seen as one final level of portfolio management (Haapala 2010). Rothman states that regardless of whether portfolio management is explicitly performed or not, it is ultimately up to the individual – whether an individual developer or a manager – to enlist the activities he or his teams are expected to work on, prioritize them and communicate this upwards to the people who are expecting the results (Rothman 2009). To help in identifying and enlisting the entire spectrum of work that needs to be attended to, Rothman (2009) proposes that in addition to "project work", there is also *periodic work*, *ongoing work* and *emergency work.* Periodic work needs to be done at a specific time but is not necessarily part of any particular project. Ongoing work is something that has to be taken care of every now and then, but attending to it is not tied to any particular time. Emergency work is something that occurs by surprise, usually as a result of some kind of crisis, and has to be attended to. Rothman recommends that ongoing work (such as checking email) should be transformed into periodic work whenever possible.

The literature reviewed provided little guidance as to how the transition of a team or an individual from multi-tasking between several development efforts to a single-backlog situation should in practice be carried out.

2.3.11 Models for agile portfolio management

This section concludes our discussion of agile software development and portfolio management. Not surprisingly, the few discovered models for agile-compatible portfolio management were based

on the notion of frequent portfolio reviews.

In addition to Shalloway's (2009) approach (already discussed in section 2.3.8 on pages 47-50 and illustrated Figure 2.14 to describe the nature of portfolio management as resource allocation across a portfolio of planned development efforts), Poppendieck and Poppendieck (2009) and Rothman (2009) provide what can be considered a series of steps to setting up agile portfolio management.

According to Rothman (2009), it is quite common for an organization to not be aware of which projects are active, which projects should be active, or which projects are planned for when. Thus, the first step is to gather the list of all activities with their supposed start and end dates. Once everything that takes up people's time has been gathered, the next step is to evaluate each activity in terms of whether it should be continued at all. The activities that survive this phase should then be prioritised against each other in a rank-ordered list, and the results of this ranking should be published along with an explanation for rationale behind the ranking. This evaluation and ranking should be made with the company mission in mind. If the company mission has not been defined or updated in a long time, this should be done before continuing. After this, actual assignment of resources to work on the activities should happen in a similar manner as to that described in (Shalloway, Beaver & Trott 2009) – see Figure 2.14 in section 2.3.8 . The evaluation and ranking of projects should be revisited at iteration boundaries, and preferably, the iterations across different activities should be synchronized.

In Poppendiecks' (2009) approach, possible development efforts that take up people's time are first classified by type, for example as *strategic business initiatives*, *feature upgrades*, *infrastructure upgrades*, and *maintenance*. Then, the desired cycle time for each type of development effort is created. The investment levels for each category are set by determining how many initiatives of each type should be carried out within a year, or in the case of activities that have no clear start or end dates (such as maintenance), a reservation of how much of the total capacity should the activity be allowed to expend is made (see Table 4 below).

Table 4 Structuring the portfolio by investment levels **(Poppendieck & Poppendieck 2009)**

Type	Timebox	Number per year
Strategic business initiative	6 months	2 of these
Business feature upgrade	2 months	12 of these
Infrastructure upgrade	12 months	1 of these
Other (e.g. maintenance)	Ongoing	20% of capaticy

Finally, the slots for the initiatives are laid out in the calendar in advance. As a time slot allocated for a certain type of initiative approaches, its actual contents are decided on based on what currently seems to be the most valuable initiative for the category in question. Thomas and Baker (2008) describe a successful implementation of Poppendieck's approach, assisted by Poppendieck in person.

Besides the experience report regarding Poppendiecks' (2008) approach by Thomas and Baker (2008), this review of the literature discovered little research or experiences on agile-compatible portfolio management. Also, perhaps because of the possible prejudice against "mixing agile and non-agile projects" (Krebs 2008), agile-compatible portfolio management remains largely

undiscussed in the context where some, but not all of the ongoing activities require the development resources' attention. This may be problematic, as such a situation may arguably be the most common one, as many, if not even most organizations developing software are somewhere along the continuum of transitioning to agile software development. In such companies, portfolio is bound to be "mixed", as only some of the teams are doing what resembles agile, others are struggling to get on board, and still others remain yet "untouched" by the transition – some of whom may even wish to even remain so.

Section 2.4 below discusses process improvement in small growing software organizations in more detail, with an emphasis on driving the improvement efforts of via the adoption of agile methods.

2.4 Adopting agile methods in small software organizations

In this section we examine the literature discovered in our review regarding software process improvement and the adoption of agile methods in small software organizations. In section 2.4.1 we outline three different types of software organizations and discuss their characteristics. Section 2.4.2 examines the growth challenges of a small organization from the perspective of managing its software development efforts. Section 2.4.3 discusses success factors for process improvement in small companies as presented in the reviewed literature and touch upon the notion of tailoring maturity models for the small software organization context. We conclude our discussion in section 2.4.4 by examining the possibility of driving process improvement in small organizations via the adoption of agile methods in the light of the characteristics of a small organization and the success factors discussed in the previous sections.

2.4.1 Types of small software organizations

Horvat et al. (2000) examine software process improvement in small software organizations, distinguishing between three different types: a small independent company, the IT department of a large enterprise, and a small branch of a larger software company.

Small, independent companies typically start with the enthusiasm of a few individuals who perform a small project for a specific customer, usually faced with challenges in terms of starting budget, insufficient equipment and so on (Horvat, Rozman & Györkös 2000). Software development in these situations is often driven by light, "strongly human-oriented" processes with "constant communication between the project members and the customer" (Pino et al. 2010a). The relationships among the personnel in such a company can be characterized by friendship and deep commitment to the company and its goals (Horvat, Rozman & Györkös 2000). Often small companies do not have enough staff to develop specialized functions or dedicate people to process improvement duties (Pino et al. 2010a). Small companies often have limited economic resources, and have little room in their budget for buying outside expertise (Pino et al. 2010a). According to Horvat (2000), the success of software process improvement in these kinds of organizations depends on the acceptance of every single employee, and human, social and cultural factors should be carefully taken into account.

According to Horvat et al. (2000), the second type of small software organization, IT department within a larger enterprise, can from a software process improvement perspective be considered as similar to the case of a small independent software company discussed above. The main difference is that its customers are other departments within the enterprise, and there may be policies affecting how these "customer relationships" are conducted.

In a small branch of a larger software company, the possible working procedures and expected, typically document-heavy inputs and outputs are often defined by the parent company (Horvat, Rozman & Györkös 2000). Thus, the efforts towards the adoption of agile methods must be able to

justify the benefits of the initiative. However, when this is done successfully, the parent company may provide the support needed for the transformation in the form of budget, equipment, facilities and training (Horvat, Rozman & Györkös 2000).

Thus, while the following sections mostly focus on the case of small, independent companies, the discussion may to a degree be applicable to all three types of small software organizations.

2.4.2 Growth challenges from the software process perspective

The vast majority of software companies reside somewhere along the continuum between the extremes of pure product or service business (Cusumano 2004, Alajoutsijärvi, Mannermaa & Tikkanen 2000). Thus, the most critical management challenge is in balancing between developing capabilities required for product business, while maintaining servicing-based business operations (Alajoutsijärvi, Mannermaa & Tikkanen 2000). This calls for simultaneously managing several businesses with differing logics (ranging from custom development, to project-wise productization via long-term customer relationships, and on to pure market-driven development), as well as forming and simultaneously managing profitable offering and customer portfolios (Nambisan 2001, Alajoutsijärvi, Mannermaa & Tikkanen 2000).

In very small (< 10 people) software companies the organizational structure is flat, and the management style is "free-flowing" and conducted by informal mechanisms and decision-making, problem resolution and communication are "based on face-to-face relationships" (Pino et al. 2010a). When a very small organization grows, individuals are usually appointed to develop software for a certain problem domain. Success remains highly dependent on the expertise of individual people (Miettinen, Mazhelis & Luoma 2010, Pino et al. 2010a, Horvat, Rozman & Györkös 2000), and the organization starts to lose its original flexibility (Larman & Vodde 2008). Small execution difficulties start to appear which are not immediately revealed by the informal management style that still was valid a very small organization (Miettinen, Mazhelis & Luoma 2010, Perry 2009, Horvat, Rozman & Györkös 2000). Projects start to running over their schedules and budgets, but slippage is measured in weeks, not in months (Perry 2009). Based on the reviewed literature it would seem that an implicit organizational structure and processes start to cause problems when an organization reaches the size of 10 employees (Miettinen, Mazhelis & Luoma 2010, Horvat, Rozman & Györkös 2000, Hofer 2002). This seems in line with the notion of the appropriate team size (5-7) as discussed in practitioner literature on agile software development, for example by Larman and Vodde (2008).

To provide financial support for growth, new projects are started in parallel with the ongoing ones, which often leads to assigning the same individuals to multiple simultaneous activities (Horvat, Rozman & Györkös 2000) sometimes working in multiple roles as well (Azar, Smith & Cordes 2007, Gilbert 2004). For example, a single person may assume the roles of project manager, business analyst, the developer, test engineer as well maintenance (Azar, Smith & Cordes 2007, Gilbert 2004). Each of these positions has an inherently different focus, which leads into unique challenges when taken on by a single person. For example, the business analyst has traditionally served as the voice of reason during project turmoil, often stopping work to further define a particular requirement, while, the project manager in the same situation remains focused on customer satisfaction (Gilbert 2004). Thus, while small growth companies view that hiring competent, specialized workforce as one of their most important challenges (Miettinen, Mazhelis & Luoma 2010) this may partly be a process issue as well: filling the shoes of a longer-term employee already playing multiple roles in too many simultaneous activities may simply be too much to ask

for from a new employee[32].

The above described scaling challenges seem reflected in a state-of-practice study of eight small to medium-sized software organizations in Eastern Finland by Saastamoinen and Tukiainen (2004) as well. The studied companies typically had five ongoing software projects, with an average team size of 3-5 employees per project[33]. All of the companies conducted "new development, maintenance and other activities", with a roughly equal effort distribution between these major types. Six of the eight companies developed custom applications for others, and five had also distinct software products developed for open markets (Saastamoinen & Tukiainen 2004).

The study identified the most important weaknesses of the organizations as being in the areas of "testing", "product management" and "project management" (Saastamoinen & Tukiainen 2004). For example, projects were sold with effort estimates based solely on the experience of the project manager, and workloads were not explicitly managed. Project plans and associated tasks were often described very roughly, and management of requirements – including changes to them – were characterized as poor and unsystematic in most of the studied organizations (Saastamoinen & Tukiainen 2004). However, interestingly enough, the studied organizations also considered their personnel's competence as a critical strength for their operations (Saastamoinen & Tukiainen 2004). Thus, it would seem plausible that the identified weaknesses would be a result of the organization's growth while the working practices have not been scaled up to current company size.

From the reviewed literature it would seem that small, software organizations are indeed experiencing growing pains. The question, as put forward by Ward (2001), remains: "Can a company manage growth-induced change in its fundamental product development processes while simultaneously maintaining enough continuity to keep the process at least minimally predictable and allow employees to plan ahead?"

The following subsections explore possible answers to this question further. First, we discuss success factors for process improvement in small organizations and touch the work on tailoring maturity models for small organizations' process improvement purposes (section 2.4.3). We conclude the discussion in section 2.4.4 by examining the reviewed literature dealing specifically with agile method adoption, and reflect the findings on the discussed process improvement success factors as well as the characteristics of small organizations.

2.4.3 Process improvement success factors for small organizations

To support the "transition of best practices" into the software industry, maturity models like ISO 15504 (also known as SPICE) and CMMI have been published and supported by different organizations (Saastamoinen & Tukiainen 2004). The awareness of such maturity models has been weak among small and medium-sized software organizations, and even if they are known and an organization recognizes its improvement needs, the models are difficult to apply because they have been designed with large organizations and government projects[34] in mind (Saastamoinen & Tukiainen 2004). While some have seen the tailoring of maturity models would as a reasonable path

[32] However, this is most likely the case in those small companies as well that do not aim for growth as well.

[33] Saastamoinen and Tukiainen (2004) do not mention whether the same individuals were assigned to multiple teams. Likewise, the article does not mention how the number of projects was calculated; we have assumed that it refers to the number of activities of the 'new development' type

[34] For example, by the U.S. Department of Defense

to pursue (Saastamoinen & Tukiainen 2004), others conclude that it is indeed very difficult to successfully apply maturity models such as CMM in small and medium software organizations (Pino, Garcia & Piattini 2008).

This review of the literature discovered several articles discussing the factors that affect the success of software process improvement efforts in small organizations. First and foremost, the improvement efforts must be directly linked to the organization's business goals, because in small organizations, the financial resources as well as the ability to absorb losses are often limited (Horvat, Rozman & Györkös 2000, Saastamoinen & Tukiainen 2004, Richardson 2002). Second, improvement efforts must produce visible benefits in a short period of time (Richardson 2002). Third, improvement efforts should focus on identifying modifications to the current ways of working instead of merely assessing the state-of-practice (Pino et al. 2010a, Richardson 2002). This – along with the previous factors – would seem to further support that applying maturity models for process improvement in small organizations may prove difficult. Fourth, as contradictory as this may seen compared to the previous success factor, the proposed modifications to the state-of-practice should as much as possible relate to existing software development models, as these can then be used as a reference for business purposes (Richardson 2002). Finally, it is suggested that process improvement efforts in small organizations should be lightweight-enough so that a quarter of the efforts of one full-time person would suffice for carrying the responsibility of directing the process improvement efforts (Pino et al. 2010b).

With these success factors and the small organization context in mind, we in the following section we examine the discovered literature that discusses the adoption of adopting agile methods.

2.4.4 Adopting agile development to drive SPI in small organizations

At least on the surface, agile software development and small software development organizations seem well matched. It would seem that small software organizations do have an affinity to adopt practices from agile software development (Hofer 2002, Taylor et al. 2008). Also, the adoption of agile methods often involves external training, which is something that has been noted to cause a "trigger effect" for process improvement in small software organizations do seem to respond to external training (Sihvonen, Savolainen & Ahonen 2006). Well-known agile methods such as Scrum are claimed to be responsible for improved lead-times, greater delivery flexibility (Taylor et al. 2008) and vast improvements in productivity and quality (Sutherland & Altman 2010) which have a high management appeal. And not only do agile methods include a built-in mechanism for identifying further improvement targets via regular reflection (Derby & Larsen 2006), but they can also be used as external references for business purposes (for example, "Our company practices Scrum"). Thus, it may seem that "adopting agile" could align well with many of the software process improvement success factors as identified in the literature.

Surprisingly, even a quite recent systematic literature review of software process improvement in small and medium software enterprises does not discuss driving SPI via the adoption of agile methods (Pino, Garcia & Piattini 2008). And even a promisingly titled article – "*Using Scrum to guide the execution of software process improvement in small organizations*" by Pino et al. (2010b) – does not actually examine how Scrum works to guide process improvement efforts, but rather, presents a process improvement approach that applies the concepts of a Backlog and Sprints to proceed in improvement efforts aiming at CMMI certification.

Indeed, organization-wide adoption of agile software development may not be entirely straightforward (Taylor et al. 2008). The proponents of agile software development tend to advocate an 'all-out-by-the-book' approach to agile adoption (Schwaber 2007), but process improvement resources and know-how in small organizations may be extremely limited (Wangenheim et al.

2006). Also, top management is likely to be more focused on dealing with imminent survival pressures (Miettinen, Mazhelis & Luoma 2010) than taking the time to 'do agile right' – which may even seem a risky makeover of the current, business-critical organizational structures and processes (Taylor et al. 2008). This notion of "taking the plunge – but carefully" is, at least according to some authors, bound to create a set of problems of its own, and from the perspective of the entire organization, having only some of the activities run using selected practices of a certain agile method may be problematic (Larman & Vodde 2010). As stated by Donald Reinertsen (2009):

> *Today's orthodoxy has institutionalized a set of internally consistent but dysfunctional beliefs. This has created a tightly interlocking and self-reinforcing system* […] *from which it is very difficult to break free. Even when we change one piece, the other pieces hold us back by blocking the benefits of change. When our change fails to produce benefits, we revert to our old approaches.* (page 3)

An examination of the literature we reviewed reveals that whether an organization should or should not go for an 'all-out-by-the-book' adoption of agile software development remains unanswered. While agile practices "should not be seen as a pick-and-mix menu from which managers can select only those they feel comfortable with" and achieving agility may require a set of interdependent practices (Vidgen & Wang 2009), others see that a slow, custom-tailored adoption of agile methods is entirely possible (Fitzgerald, Hartnett & Conboy 2006). As another example of the prevailing contradictions, Lawrence and Yslas (2006) state that adopting all of the practices in eXtreme Programming is neither required or necessary. However, it is questionable whether the deviations from by-the-book-XP described by Lawrence and Yslas (2006) – such as not doing pair programming, using headphones to block out team room noise and "having teams that are not composed of only senior developers" – actually are in contrast with the principles of agile software development.

Thus, whether a small software organization should or should not "take the plunge" of an all-out agile adoption, it would based on the reviewed literature seem quite plausible that most small software organizations who are interested in adopting agile software methods will find themselves in a little-described situation where some – and only some of their activities would be conducted, or at least would be striving towards agile software development. Also, from the perspective of a small company intent on growing, the single project/single customer view of agile methods does little to help the link the development process with the business decision-making of the company. While product and portfolio management processes would not have to be as robust as in larger companies, they still need to provide adequate structure to leverage agile methods' potential benefits, or at least, prevent those activities that employ agile software development from harming those activities which are conducted in a 'traditional' manner.

2.5 Summary

In sections 2.2 and 2.3 above, we reviewed existing work on software product and portfolio management as seen through the lenses of "traditional" new product development and product management literature, software engineering research, as well recent experience reports and practitioner literature on agile software development. In addition, we in section 2.4 discussed driving process improvement in small organizations via the adoption of agile methods in the light of the existing literature.

The reviewed literature on software product management and management of new product development tend to view development as an activity that can be planned in detail and then executed according to the plan, which is very much in contrast with the mindset of agile software development. Perhaps as a result, the reviewed literature also remains conceptually disconnected from agile software development. Furthermore, the models and theory for product and portfolio management are largely derived from a large company context, and thus their direct applicability to

the context of small organizations and teams developing software is questionable.

Most of the literature on agile software development on the other hand has focused on the context of a single team working on a single backlog concerning a single product. Concerns regarding long-term product and release planning and portfolio management have until very recently been largely swept under the proverbial rug of the product owner role, who has to single-handedly deal with the most difficult questions involved in software development, like deciding what exactly should be developed, and in which order. Thus, "out-of-the-box" agile software development seems to link poorly with product and portfolio management poorly as well.

Combining advice from recent practitioner literature and experience reports on agile software development as well as requirements engineering research, we were able to outline on a conceptual level how roadmapping, release planning and portfolio management can be understood and linked with agile software development. The key elements for the proposed link were *planning levels*, *progressive refinement of work items*, *hierarchical work item/goal structures*, as well as *viewing portfolio decision-making as occurring on multiple levels*.

While agile software development and small software development organizations seem on the surface well matched, organization-wide adoption of lean/agile may not be entirely straightforward. It is plausible that most small software organizations interested in adopting agile software methods would find themselves in a little-described situation where some – and only some of their activities would be conducted, or at least would be striving towards agile software development. Thus, product and portfolio management processes have an important role as they are needed not only to provide adequate structure to support the growth of a small organization, but also to leverage agile methods' potential benefits while doing so, and help the "traditionally managed" and "agile" activities co-exist in an organization's development portfolio.

In conclusion, there is ample room for research in understanding how agile software development should be linked with product and portfolio management, and especially in the context of small organizations that strive for growth. Thus, we continue on to chapter 3 (*Research problem and* methods) where we from this background pose the specific research problem and the research questions this dissertation aims to answer.

3 Research problem and methods

In this chapter, the research problem and the research questions are posed and the rationale behind them is explained (section 3.1). Then, the research approach is described, along with listing the case companies discussed in the included publications (section 3.2). Then we describe the specific steps undertook for answering each research question along with the research methods used (section 3.3).

3.1 Research problem and research questions

Based on the discussion of the related work (see summary in section 2.5 on page 56), the research problem of this study is posed as follows:

How can product and portfolio management be linked with agile software development?

The research problem is addressed through answering the following research questions. The research questions, as well as how they follow from one another are further explained in the paragraphs that follow.

RQ1. What are the key processes and decision areas that connect Business with Iterative and incremental software development?

RQ2. What does roadmapping mean in the context of agile software development and how can a roadmap be visualized?

RQ3. Do small software organizations suffer from the lack of explicit portfolio management?

RQ4. How can portfolio management be set up and conducted in small software organizations striving for agile software development?

RQ5. How can long-term product and business planning be linked to daily software development tasks in agile software development? What role does portfolio management play here?

RQ6. What is known in the existing literature about a) product and portfolio management in the context of agile software development, or b) the enterprise-wide adoption of agile software development in small organizations?

The relationships of the research problem and the research questions to the publications are illustrated in Figure 3.1 and explained below.

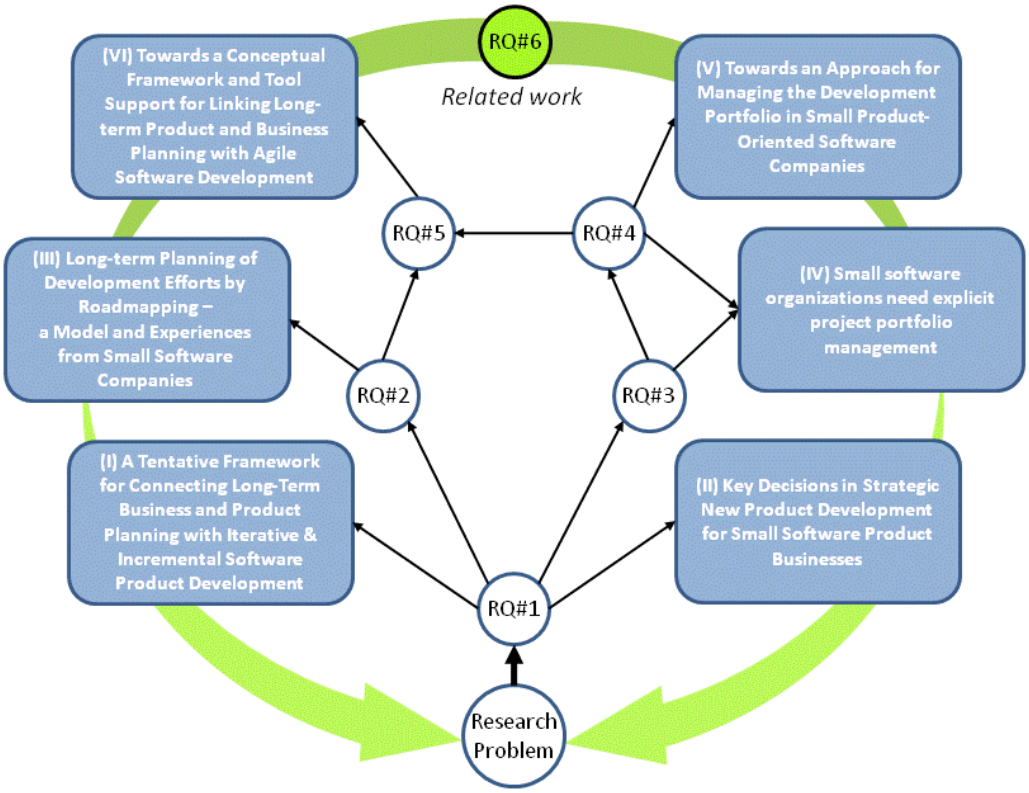


Figure 3.1 The research questions' relationships to the publications and each other

The discussion of the related work in chapter 2 identifies *release planning*, *product planning/roadmapping* and *portfolio management* as the key processes that should be understood in a new light if the benefits from agile software development are to be utilized by the entire organization. However, most of the work referenced there was not published at the time the research for answering RQ1 was carried out and published (publications I and II). Because of this, RQ1 includes the question of what the key processes are. The rest of the research questions delve deeper into understanding the mentioned processes themselves, with RQ2 and RQ3 leading into RQ5 and RQ4, respectively. The answer to RQ6 sets the answers to the other research questions into context by an in-depth examination of the related work. Figure 3.1 illustrates the "flow" of questioning with the black narrow arrows. This is also further explained below.

As discussed in section 2.2.2, release planning has attracted attention among software engineering researchers, but much of the proposed models are difficult to apply without understanding their real-world context in terms of the roadmapping, portfolio management, as well as agile software development processes. As even the big picture of how the portfolio management, release planning and roadmapping processes should in agile software development interact has not been established, developing models for release planning or examining the proposed algorithmic release planning closer is not in the scope of this dissertation. Sound research regarding roadmapping seems to be emerging but a practical example, as well as understanding how roadmapping relates to agile software development is missing. This is addressed by research questions RQ2 (publication III) as well as RQ6 (the systematic literature review in chapter 2).

Of the three key processes, release planning, roadmapping and portfolio management, the latter has

so far – at least in the context of agile software development – been least addressed by other researchers and authors. Thus, two research questions (RQ3 and RQ4) have been dedicated to address the nature and practice of portfolio management (publications IV and V).

Research question RQ5 ties together the answers to the other research questions by describing the role of the portfolio management process in linking the long-term product and business plans expressed in roadmaps with the daily software development in the context of agile methods (publication VI).

Research question RQ6 relates the results of the included publications to their larger context in the field of software engineering and management of new product development, and where appropriate, serves to complement our results the more recent insights from other researchers and authors.

3.2 Research approach and case companies

Information systems and software engineering research have been criticized as disconnected from practice and producing results that from a practitioner perspective are often nearly irrelevant (Davison, Martinsons & Kock 2004, Glass 2009, dos Santos & Travis 2009, Baskerville & Myers 2004, Peffers et al. 2007). Likewise, the field of organization and management studies is often criticized as fragmented and of limited practical relevance (Denyer, Tranfield & van Aken 2008). In contrast, the research problem of this dissertation is a prime example of a complex topic that cannot be studied outside of its practical context. Thus, the approach taken in this thesis is that of design science (Peffers et al. 2007, Järvinen 2007, Aken 2004) and constructive research (Kasanen, Lukka & Siitonen 1993, Kekäle 2001). The constructions are based on the findings and lessons learned from qualitative (Patton 2002) case studies (Yin 1994, Runeson & Höst 2009) conducted as action research (Davison, Martinsons & Kock 2004), as well as synthesis of related work (Denyer, Tranfield & van Aken 2008). As evident from the explanation of the research problem and research questions in section 3.1, several rounds of the industry-as-laboratory (Potts 1993) research life-cycle (Flint 2009) were underwent as the focus of this study on linking the key business decision processes of roadmapping, release planning and portfolio management to modern, agile software development processes gradually became clearer.

Table 5 below provides an overview of the 14 case organizations studied in the included publications. The case organizations have between the year 2000 and present participated in the SEMS, SHAPE and/or ATMAN research projects at the Helsinki University of Technology (Aalto University from the beginning of 2010). These research projects have been funded by the companies (20-25%) and the Finnish National Technology Agency Tekes (75-80%). Thus, the sampling of the case organizations is purposeful; the implications of this on the results are discussed in limitations, see chapters 4 (*Results*) and 5 (*Discussion*).

While some of the case companies, for example, *Hephaestus, Janus, Mercury* and *Cheops* are medium-sized, all of the studied software organizations are small (that is, have under 50 employees). *Proteus* and *Theseus* are independent business units of medium-sized companies, both of which have roughly 100 employees. In Table 5, the columns *# of employees* and *business / offering* refer to the total number of people in the company (or independent business unit) at the time the research for the related publication(s) was conducted.

Table 5 The case companies or business units studied in the included publications

Pseudonym	Em-ploy-ees	Business / offering	Publications featured in	Total period of co-operation
Achilles	19	Mobile enterprise solutions and professional services	IV, V	2003-2004
Ajax	15	Solutions and services for automating safety-critical logistics	IV, V, (VI)	2004-2010
Hector	21	Solutions and services for relationship and customer information management	IV, V, VI	2003-2004, 2007-
Odysseus	15	Solutions for securing electronic transactions	IV, V	2003-2004
Proteus	20	Systems for optimizing the operation of transport vehicles	IV, (VI)	2003-2010
Theseus	40	Solutions and services for industrial business integration	IV, (VI)	2007-2010
ToolCo	14	Toolkit for user interface development and custom software development services	III	2001-2002
TeamCo	40	A platform and applications for mobile social media	III	2001-2002
MobAppsCo	40	Solutions for mobile productivity	III	2001-2002
Mercury	100	Web-based entertainment portal	VI	2007-2010
Hephaestus	400	Solutions for construction modelling	I , (VI)	2004-2010
Janus	400	Solutions for computer security	VI	2007-
Slipstream	30	Solutions and services for streaming mobile video	II	2003-2004
Cielago	30	Wireless module and access device solutions	II	2002-2003
Cheops	60	Solutions and services for business process modelling	II	2002-2003

3.3 Methods

Table 6 below provides an overview of the research methods used for answering the research questions as well as the related publications.

Table 6 Mapping research questions to the methods used and the included publications

Research question	Research methods used					Publications
	Literature study	Design research	Case study	Action research	Survey	
RQ1: What are the key processes and decision areas that connect Business with Iterative and incremental software development?	**X**	**X**	**X**	**X**		**I, II**
RQ2: What does roadmapping mean in the context of agile software development and how can a roadmap be visualized?	**X**	**X**	**X**	**X**		**III**
RQ3: Do small software organizations suffer from the lack of explicit portfolio management?	**X**		**X**		**X**	**IV**
RQ4: How can portfolio management be set up and conducted in small software organizations striving for agile software development?	**X**	**X**	**X**	**X**		**IV,** V
RQ5: How can long-term product and business planning be linked to daily software development tasks in agile software development? What role does portfolio management play here?	**X**	**X**	**X**	**X**	**X**	**VI**
RQ6: What is known in the existing literature about a) product and portfolio management in the context of agile software development, or b) the enterprise-wide adoption of agile software development in small organizations?	**X**					**Chapter** 2

The research process and the methods used for answering each research question are explained in more detail in the sections that follow (3.3.1-3.3.5). Additional detail can be found from the included publications themselves. Selected interview templates can be found from Appendix B of this summary. Unless otherwise mentioned, all of the interviews mentioned in the sections below

have been tape recorded.

3.3.1 Approach and methods for answering RQ1

The steps taken to answer RQ1 (*What are the key processes and decision areas that connect Business with Iterative and incremental software development?*) were:

1) Undertaking a literature study to a) uncover existing work in the area and b) understand the characteristics of small companies as well as the software business
2) Adapting an existing framework that discussed key product development decisions to comply with the characteristics of small companies and the software business
3) Further developing and validating the framework by using it to analyse three small software companies for the purposes of providing them with suggestions to improve their processes
4) Adapting another existing framework – the Cycles of Control – to distinguish the key decision areas of *portfolio management* and *product management* as two distinct processes that play a key role in linking business and development decision-making
5) Refining and validating the resulting framework by using it to analyse the current state, problems and challenges of a larger software company with a successful 20+ year history in separate sessions with three experienced VP level managers from the same company

These steps are illustrated in Figure 3.2 and explained in more detail further below.

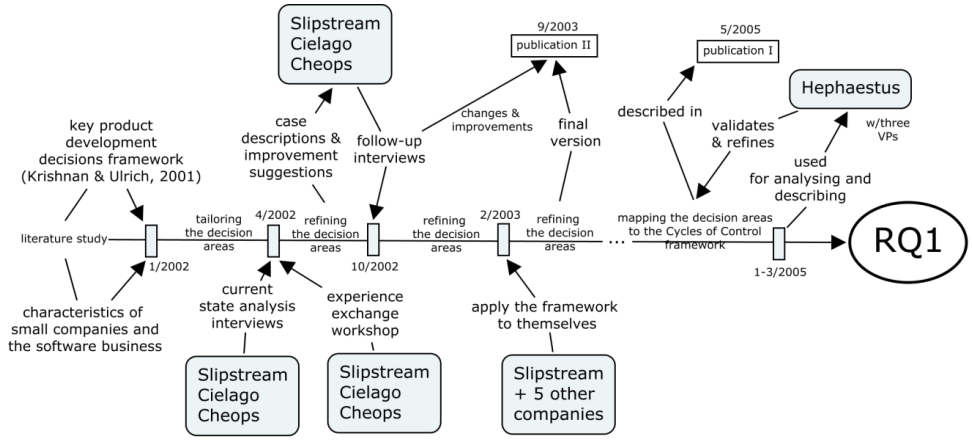


Figure 3.2 Research conducted and published to answer RQ1 on a timeline

To find existing work discussing the links between business and development decision-making, we undertook a review of the literature on strategic management, management of new product development and software engineering. The areas of literature were chosen because they all to at least some degree address the management of new product development from different perspectives. This review of the literature did not follow a defined protocol.

Our search did not uncover advice specific to small software organizations. Thus, we chose a generic framework on product development decisions (Krishnan & Ulrich 2001) with the intent of tailoring it to the small software company context. The first round of tailoring the key decisions framework was done based on what information we in the same review found from the literature regarding the nature of the software business as well as characteristics of small companies.

The validity of our initial 'key decision areas' for the small software product business context was

evaluated by examining whether describing real case companies' practices and processes would be meaningful using such a structure (see Appendix B, Figure B1), and whether relevant problems and challenges could be found using the decision areas as a checklist. For this, we selected three small software companies from among our research project's industrial partners at the time based on their willingness to participate in jointly examining how they conducted long-term planning of product development efforts and how they could improve on the current practices.

Semi-structured interviews (Patton 2002) were conducted at each company to understand how they dealt with the respective decision areas and the related software engineering practices. The interviews were tape recorded. In all three of the companies, called Slipstream, Cielago and Cheops (having 20, 10 and 30 development people at the time of the interviews, respectively), the head of product development was interviewed. At Cheops, we additionally interviewed one project manager and two product managers because of the more extensive partnership deal the company had subscribed for in the ongoing collaboration. The data from the interviews was thematically coded (Miles & Huberman 1994) using the elements of our initial key decision areas. Where necessary, decision areas (or parts thereof) were iteratively added, renamed and restructured to better address the issues of importance that surfaced during the interviews.

Case descriptions structured according to the decision areas illustrating the companies' practices and highlighting their strengths, weaknesses, problems and challenges were written, checked for correctness with company representatives, and presented to the companies along with improvement suggestions. These suggestions were based on contrasting the companies' current practices (or the lack thereof) with the areas outlined by our framework. The general findings were presented to all of the companies in a joint workshop. After this, company-specific observations and improvement suggestions were discussed individually at the companies in three-hour workshops, and soon thereafter, we conducted informal discussion-like interviews with the interviewed persons to evaluate how useful they perceived the analysis and the suggestions made.

After an observation period of four to six months, a second round of semi-structured interviews (Patton 2002) was conducted to find out what action, if any, had resulted from the first interviews and their dissemination. At each case company, the head of product development was interviewed. Additionally, at Cheops, one product manager, a project manager, a developer and the head of quality assurance were interviewed as well. From the perspective of the key decisions framework, the case studies resulted in adding one-fourth of the total content present in current version, and led to a make-over of the structure of the decision areas as well as much of the terminology used.

Four months after the final interviews, we arranged an experience exchange workshop in which six small software companies (including one of the original case companies) presented their current product development practices, and their self-perceived strengths and challenges using the structure of the framework. From the perspective of the framework, the workshop resulted in some structural changes and terminology changes to simplify the framework and make it more intuitive. The interview questions, detailed case descriptions and a step-by-step description of how the framework evolved during the course of the study can be found from the appendixes of (Vähäniitty 2003). The results so far were summarized in publication II.

To fully answer RQ1, the overall picture of how the two most important decision areas, portfolio management and product management should connect to both business and development decision-making was done by examining them through the lens of time pacing (Gersick 1994) using the Cycles of Control framework (Rautiainen 2004). The resulting framework was refined and validated by reflecting it with the processes, practices and perceived improvement needs of Hephaestus, a larger company with 400 employees and three separate business units. We conducted unstructured, conversation-like interviews (see Appendix B, Figure B2) with three of the

company's vice presidents (two from business units and one responsible for processes) in three separate sessions. This company was chosen because of its involvement in our research projects at the time, but also because two of its business units corresponded to small independent software development organizations, while one of the units was larger (200 people), providing the opportunity to examine possible differences. The refined framework and the lessons learned were summarized in publication I.

The results corresponding to RQ1 are presented in section 4.1 (*Connecting business and software development*) in chapter 4 (*Results*).

3.3.2 Approach and methods for answering RQ2

The steps taken to answer RQ2 (*What does roadmapping mean in the context of agile software development and how can a roadmap be visualized?*) were:

1) Interviewing the managers and developers of a small software company interested in roadmapping with respect to their current development practices as well as their perceived needs for roadmapping
2) Developing an initial notation for visualizing product development and servicing plans of the case company based on an existing notation
3) Iteratively refining the notation as the CEO of the company used it over a period of four months to depict his plans for developing and offering a new product concept
4) Reflecting on the resulting roadmapping visualization with other two other small companies who were interested in roadmapping, and comparing it to their current practices
5) Conducting a retrospective in-depth interview with the CEO who was in charge of the roadmapping process on the value of the exercise
6) Review of the literature to uncover the work concerning software product roadmapping published between 2002 -2009

These steps are illustrated in Figure 3.3 and explained in more detail further below.

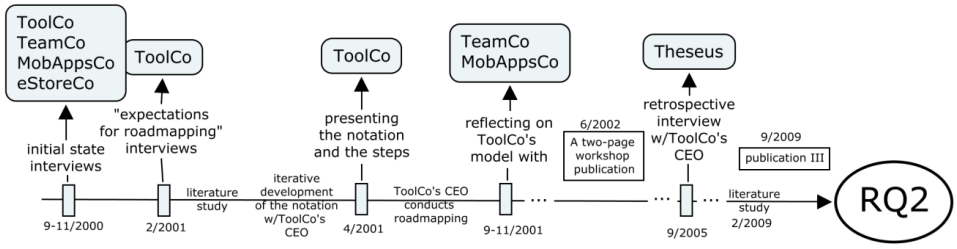


Figure 3.3 Research conducted and published to answer RQ2 on a timeline

In 2000, we conducted so-called initial state interviews at the four companies involved in our research project at the time. These interviews were semi-structured (Patton 2002). All of the companies were small, and at each company we interviewed between 4-6 people, including both top management and developers. At ToolCo, the initial state interviews revealed that the company had envisioned a product concept based on its software toolkit for rapid creation of browser-enabled user interfaces. Putting together a toolkit to help in performing the project work, the main source of revenue at the time, had been a conscious effort since the founding of the company in 1996. However, the toolkit's commercialization was a more recent idea and had surfaced during 1999.

The managers at ToolCo were interested in conducting "roadmapping" for the envisioned product concept, by which they meant the planning of the contents of the upcoming releases, possible

related services, and the resourcing implications of these. They asked us to help in figuring out based on existing literature how roadmapping should be done, who should participate, what kind of visualizations (if any) should be used, what should be included in the roadmap, and what should be considered when conducting "roadmapping". Helping ToolCo with their roadmapping was taken up by the author together with three other students of Industrial Engineering and Management from the Helsinki University of Technology, Suvi Elonen, Lassi Lindblom and Ilkka Miettinen. In preparation for the literature review we performed unstructured, conversation-like interviews three more people to better understand the problems ToolCo wanted to tackle with "roadmapping": the CEO, the manager in charge of the development process, and a senior developer.

Over the course of the following months, we iteratively developed and refined a notation for visualising product roadmaps in co-operation with ToolCo's CEO. The source that most influenced the initial version of our notation was the notation reproduced in an article by Wells et al. (2004), but other sources were also used. We presented our "final notation" and accompanying process steps for creating and updating roadmaps to the people at ToolCo in spring 2001. After this, the author conducted periodical unstructured, conversation-like interviews with the CEO of the company as he practiced roadmapping over a period of four months to depict and refine the plans for developing and offering the new product concept. The feedback on the visualization from these interviews resulted in several changes, with the most important ones being the introduction of a *service layer*, including explicit *resource types*, and simplifying the notations for minor releases and release composition.

In the fall of 2001, discussions were opened with TeamCo and MobAppsCo to reflect on the work conducted with ToolCo and compare it to the companies' current practices in the area. At MobAppsCo, our study was intentionally limited to discussing our model and the company's practices in the area of release and product planning. At TeamCo, the author prepared an example roadmap to demonstrate the use of the visualization based on TeamCo's current plans and discussed the result with the involved managers. The visualization seemed well received by TeamCo's people. However, neither these visualized plans, nor any other plans regarding future releases expressed at that point were subsequently followed, as TeamCo's key development resources were caught up servicing the current customers (for example installing the system, doing systems integration, customer-specific tailoring, consulting and training). Six months later, TeamCo filed for bankruptcy.

In 6/2002, a two-page workshop paper summarizing the results so far was published (Vähäniitty, Lassenius & Rautiainen 2002).

In 2005, the author conducted one more semi-structured interview (Patton 2002) with the CEO of ToolCo who had been in charge of the roadmapping effort with the intent of a retrospective evaluation of the cost and benefits from performing the roadmapping exercise back in 2001. However, as the interviewee was now employed at Theseus (another case company studied in the publications included in this dissertation; see section 3.2) the interview ended up mostly discussing the current and envisioned practices of regarding the long-term planning of development efforts and portfolio management at Theseus. Thus, the results from that interview are not included in publication III but instead serve as background information for the problem setting of publication IV.

Publication III (9/2009) is an expanded version of the 2002 workshop paper and is based on the empirical data from 2000-2001. For the preparation of the expanded description of the roadmapping approach and visualization as publication III, the author conducted, together with a colleague, Pasi Pekkanen, an unsystematic literature review of the work on software product roadmapping published between 2002 and 2009.

The results corresponding to RQ2 are presented in section 4.2 (*Roadmapping*) in chapter 4 (*Results*)

3.3.3 Approach and methods for answering RQ3

The steps taken to answer RQ3 (*Do small software organizations suffer from the lack of explicit portfolio management?*) were:

1) Selecting three small software companies and conducting a current state analysis of the work practices and perceived problems and challenges regarding the case companies' software development activities
2) Conducting a systematic literature review to identify what problems are associated with inadequate portfolio management
3) Revising the interview questions to include topics that directly address whether portfolio management is being conducted and whether the problems identified in step 2) are present
4) Selecting three more small software development organizations for studying their work practices and perceived problems and challenges using the revised question set
5) Synthesising a list of problems common to all of the case organizations, and comparing this to the list of problems associated with inadequate portfolio management

These steps are illustrated in Figure 3.4 and explained in more detail further below.

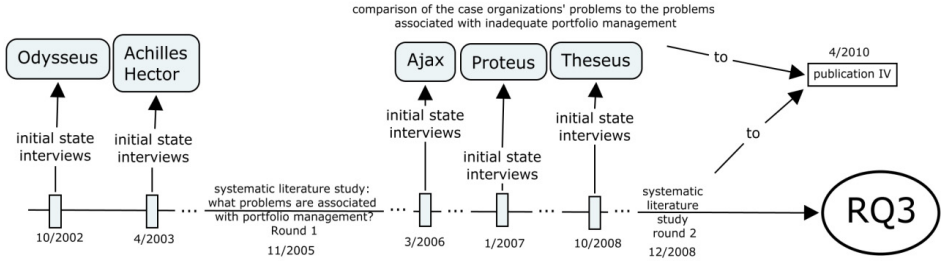


Figure 3.4 Research conducted and published to answer RQ3 on a timeline

As part of the standard practice in our research projects when starting the work with a new partner company, we have conducted a current state analysis. In the case of these companies, this meant conducting semi-structured (Patton 2002) in-depths interviews with the product development managers of Achilles, Hector, and Odysseus to gain an overview of the companies' work practices and perceived problems and challenges. These semi-structured interviews (see Appendix B, Figure B3) were followed by informal conversation-like interviews with other product development personnel to gain additional perspective.

Based on the interviews, we started to suspect that many of the perceived problems small software companies are facing could in fact stem from inadequate portfolio management. To further examine this, we conducted a literature review to identify what problems are associated with of inadequate portfolio management. The literature review was conducted in two steps. First, we manually looked through 19 books on managing new product development and/or software development in order to provide a preliminary outline of the symptoms associated with inadequate portfolio management and to identify keywords for database searches. See publication IV as well as Vähäniitty (2006) for details on the keywords and testing for keyword validity. These books were informally selected, based on their accessibility and perceived relevance. We then continued with a systematic review (Brereton et al. 2007) through the ScienceDirect™ portal, thus covering the journals that are considered relevant for research on portfolio management (Artto et al. 2009) as defined in

publication IV. A first round of the database searches was conducted in 11/2005, discovering a total of 26 relevant research papers that contained material describing inadequate portfolio management and/or the typical problems that occur in conjunction with it.

The overlap of the found problem lists was limited, warranting the creation of a synthesized list of problems, which we did. Based on our synthesized problem list, we revised our interview structure to include questions that directly addressed whether portfolio management was being conducted, implicitly or explicitly, and whether the personnel experienced symptoms that, in the literature, are associated with inadequate portfolio management.

We then approached the rest of the small software development organizations that we at the time were co-operating with: Ajax, Proteus, and Theseus. For these case organizations, all of the interviews were semi-structured (Patton 2002), and the revised interview questions were used. We interviewed five to ten people at each of the organizations. As a representative example of the kind of individuals we sought to interview, the eight interviewees at Theseus were a business analyst, a business development person, a business unit head, a process manager, a system architect, a system developer, and two project managers. Theseus was is an independent business unit of a 100-person company, while the other case organizations in publication IV are small companies with 7–30 developers and 15–40 personnel in total (see Table 5 on p. 61). The possible implications of the selection of companies on the results are discussed in the limitations section regarding the answer to RQ3 (section 4.3.3).

The final step in answering RQ3 was to compare the problems experienced by the case organizations to the problems that in the literature are associated with inadequate portfolio management.

For the sake of completeness, we conducted a second round of literature review in 12/2008 to uncover the work published after the first round in 11/2005. In the second round, we used the same protocol and 8 new relevant research papers were discovered. While they added detail, this did not lead to significant changes in our synthesized problem list.

The results corresponding to RQ3 are presented in section 4.3 (*Do small software organizations need portfolio management?*) of chapter 4 (*Results*).

3.3.4 Approach and methods for answering RQ4

The steps taken to answer RQ4 (*How can portfolio management be set up and conducted in small software organizations striving for agile software development?*) were:

1) Initial state interviews at Achilles, Hector and Odysseus – see step 1) in section 3.3.3 above
2) Observing meetings that dealt with portfolio level decision-making at Hector, and monthly interviews of the product development managers at Achilles and Odysseus regarding their portfolio management practices
3) Conducting a literature review to discover methods, models and techniques for portfolio management; conducted in parallel with step 2)
4) Identifying key challenges, discussing possible ways to improve the practices, taking selected actions with the company personnel, and observing the results at each case company; conducted in parallel with step 2)
5) Outlining an approach for setting up portfolio management in small software organizations; conducted in parallel with step 2)
6) Conducting a final state interview at each company to summarize the key changes in ways of working and benefits as perceived by the interviewees
7) Complementing the approach for setting up portfolio management with the portfolio

management practices employed at Theseus

These steps are illustrated in Figure 3.5 and explained in more detail further below.

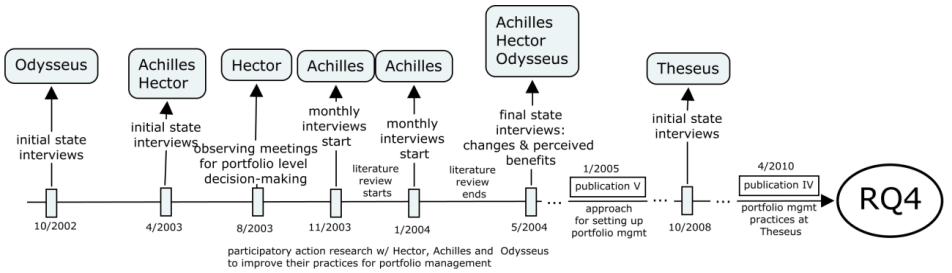


Figure 3.5 Research conducted and published to answer RQ4 on a timeline

The first step taken to answer RQ4 refers to the interviews with the product development managers and informal discussions with development personnel at Achilles, Hector and Odysseus. This is the same as the first step taken to answer RQ3 described in section 3.3.3 above, and is explained there in more detail.

The second step was to find out what the state of practice of portfolio management in the case companies was. For this, we examined more closely how the case companies' product development portfolios and long-term product development plans were managed, and what the problems and challenges involved were. From August 2003, Hector let us attend as observers in portfolio management meetings that ranged from 2 to 3 hours each. We took notes, and after each meeting asked for necessary clarifications and discussed possible insights with the head of product development, who also attended the meetings. This helped us in understanding the nature of portfolio management in small software organizations as well as finding the right questions to ask at Achilles and Odysseus, where direct observation in portfolio level decision-making meetings was not possible.

At Achilles and Odysseus, we conducted structured interviews (Patton 2002) of 1-2 hours in length on a monthly basis. The interviews were tape recorded. The interviewed persons were responsible for managing product development. In each meeting we asked how the companies were conducting portfolio management, the justification for possible changes to the process, as well as the currently perceived problems and challenges. This interview period started in November 2003 at Achilles, and in January 2004 at Odysseus.

In parallel to conducting the monthly interviews and observation, we conducted a review of literature to discover methods, models and techniques for portfolio management. We discussed our findings with key personnel in the companies in order to understand the practical applicability of existing models and practices. This gave us additional insight into how the companies saw their portfolio management process and helped us identify some limitations of existing research.

To understand how the gaps between the existing literature and the kind of advice the companies' needed could be overcome, we outlined a preliminary approach for implementing portfolio management in small, product-oriented software companies. This served as a rough roadmap for improvement work in the case companies. Together with key personnel in each company, we identified courses of action to take to improve the portfolio management process. Each month we, together with the companies' representatives, evaluated the progress of the improvement work, identified new challenges and courses of action to take, and refined our approach to setting up portfolio management to reflect the insights and lessons learned.

In May 2004 we conducted in-depth interviews with the product development managers of each company to obtain a snapshot of the state of the portfolio management process, the current composition of the development portfolios, and the perceived benefits from the improvement work so far.

The final step refers to the inclusion of the practices employed at Theseus (from publication IV) as a part of the answer to RQ4, as these provide an independent example (that is, independent of the influence by the author) of what explicit portfolio management in a small software organization can consist of.

The results corresponding to RQ4 are presented in section 4.4 (*Portfolio management for small software organizations*) in chapter 4 (*Results*).

3.3.5 Approach and methods for answering RQ5

The steps taken to answer RQ5 (*How can long-term product and business planning be linked to daily software development tasks in agile software development? What role does portfolio management play here?*) were:

1) Adding detail to the conceptual framework presented in publication I to describe more closely how long-term product and business planning can in theory be linked to daily software development tasks in agile software development and the role of portfolio management there
2) Developing a software tool (Agilefant) to help evaluate the validity of the framework in practice
3) Obtaining initial validation and input to further refine the framework by using it to identify targets for process improvements at two software companies (Hephaestus and Mercury)
4) Weak market test part I: adopting Agilefant to manage all activities at Hector (a small software company)
5) Weak market test part II: surveying for and estimating the size of Agilefant's world-wide user base, and looking up reviews that compare Agilefant to other tools for agile project management
6) Adapting and improving the conceptual framework and Agilefant to accommodate insights from new literature and research since publication VI, as well as the practical experiences of the author and his colleagues from using Agilefant to manage our own 'portfolio'
7) Weak market test part III: interviewing a product owner from Janus who applied the hierarchical system of goals prescribed by the framework to plan an upcoming release

These steps are illustrated in Figure 3.6 and explained in more detail further below.

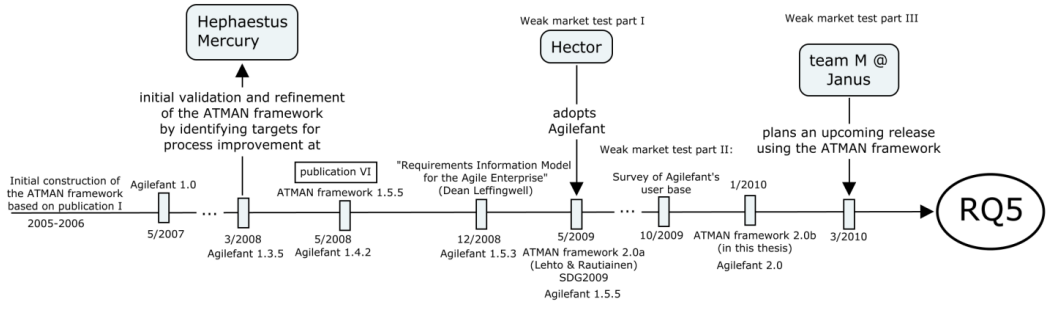


Figure 3.6 Research conducted and published to answer RQ5 on a timeline

After publication I, the author started, based on the results from publications II, V as well as the data collected for publication III to work on a more detailed description of how long-term planning and plans should be linked to the developers' daily work in the context of a development portfolio containing both 'agile' as well as 'non-agile' activities that required the developers' attention. This framework is referred to here as *the ATMAN framework*, according to the ATMAN (Approach and Tool support for development portfolio MANagement) research project[35] at Helsinki University of Technology (also known as Aalto University, School of Science and Technology since the beginning of 2010).

Shortly after an initial version of the framework was constructed, the author and a colleague, Juha Itkonen, began to direct the efforts of a group of students at the Helsinki University of Technology to develop a software tool that would support an organization that wanted to manage its activities along the lines directed by the framework. The basic idea in developing a support tool was to validate and further refine the framework, as well as to help deploy the framework in industrial settings and refine it based on the lessons learned. This notion of using a support tool – Agilefant – to establish a symbiotic relationship between research and the software industry is illustrated in Figure 3.7 below. If an organization would successfully adopt the tool, use it, and perceive it as beneficial, this would both achieve a degree of validation for the framework, as well as feedback to further improve the framework as well as the tool itself.

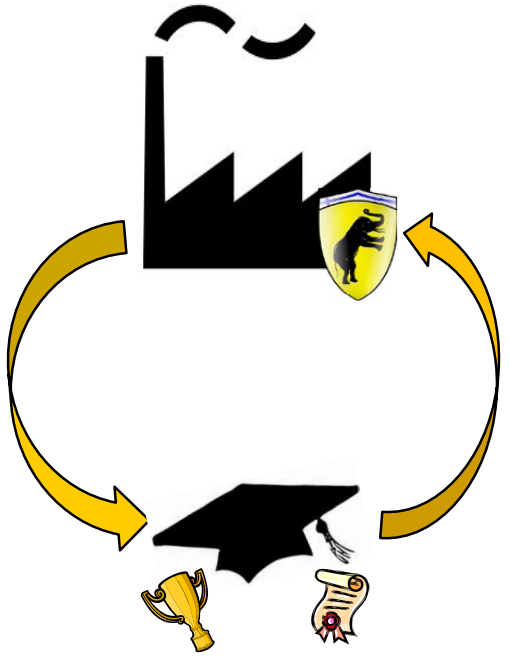


Figure 3.7 Establishing a symbiotic relationship between industry and research with Agilefant

The first version of Agilefant was released in May 2007 (see Figure 3.6). Agilefant was further developed by research assistants Ilkka Lehto and Ville Heikkilä (who subsequently became as part of the ATMAN research project), and in September 2007 Agilefant was mature enough to be

[35] See http://www.tekes.fi/ohjelmat/Verso/Projektit?id=9455127 and http://www.soberit.hut.fi/sprg/projects/atman/

adopted to manage the author's and his colleagues' research and teaching work. The development of the conceptual framework, as well as that of Agilefant, further continued in 2007-2008, again by a team of students, with the author coordinating the efforts. A version of the framework was in March 2008 used by the author and his colleagues for directing process improvement efforts at two case companies, Janus and Mercury. Using the framework we were able to identify missing responsibilities, decision-making structures, and poorly defined roles, as well as propose tangible improvement suggestions which the companies acted on. These efforts also served to further validate and refine the framework, and publication VI contains the resulting version (1.5.5).

Mainly powered by the efforts of student groups as well as research assistants Reko Jokelainen, Pasi Pekkanen and Antti Haapala, the development of Agilefant continued towards the capabilities prescribed by the framework. In December 2009, Hector's CEO approached the author with the intent of rejuvenating the more intensive mode of research collaboration from 2003-2004. The CEO's intent was to seek the author's help in a company-wide adoption of the latest version of Agilefant, as well to further tailor Agilefant to suit Hector's needs. In February 2009 Hector started to pilot Agilefant in its most important ongoing project at the time, and by May 2009, all the work of Hector's employees was accounted for in Agilefant. Coincidentally, by that time, Agilefant had also reached the capabilities directed by the version of the framework described in publication VI.

Further evidence of how well the framework from publication VI and the respective tool support, Agilefant, pass the weak market test was sought via the first annual Agilefant user survey in November 2009. In addition to posting an invitation to answer the survey on the forum and to the news sections at Agilefant.org, an email invitation was sent out to everyone who either had

a) non-anonymously posted on the Agilefant forum,
b) inquired about Agilefant by email
c) registered into Agilefant's publicly accessible issue tracking system,
d) non-anonymously edited the Big List of Tools[36] at Agilefant.org, or
e) participated in Agilefant's development between 2006 and present

To complement the survey, we also searched for reviews of open source and/or free tools for managing agile software development. One relatively recent such review was found (see section 4.5 for a discussion of its contents).

In March 2010, the author interviewed the product owner of a relatively independent ten-person team at Janus regarding his efforts of planning an upcoming major release using a system of hierarchical goals as prescribed by the ATMAN framework. The interview was conducted in two parts: first, to get an overview of to what extent the product owner had independently created a so-called 'story tree' for the upcoming release. At the end of this first interview, the author asked whether the product owner would be interested in fully organizing the story tree for the product according to the guidelines provided by ATMAN framework, and whether he would see the value in it. The product owner agreed, and the second part of the interview, where the re-structured story tree was examined and discussed was scheduled and conducted two days after the initial interview.

With respect to the further development of the framework itself, the two most important influences to the version of the framework (referred to as 2.0b in Figure 3.6) presented in the summary part of this dissertation (see section 4.5 *Linking long-term planning and daily work*) were the "Lean and scalable requirements model for the agile enterprise" by Dean Leffingwell (Leffingwell 2011)[37] and

[36] http://tinyurl.com/biglistoftools

[37] To our knowledge, originally published 12/2008 in a blog post at http://www.scalingsoftwareagility.com

the development of the 2.0(a) version of the ATMAN framework (Lehto & Rautiainen 2009, Lehto 2010) by the author's research team.

The results corresponding to RQ5 are presented in section 4.5 (*Linking long-term planning and daily work*) in chapter 4 (*Results*)

3.3.6 Approach and methods for answering RQ6

The results of a particular research study – or even a set of studies such as the included publications – cannot be interpreted with confidence unless they have been considered together with the results of other studies addressing the same or similar questions (Cruzes & Dybå 2011). Thus, for answering RQ6 (*What is known in the existing literature about a) product and portfolio management in the context of agile software development, or b) the enterprise-wide adoption of agile software development in small organizations?*), we conducted a systematic review of the literature to produce a narrative synthesis (Petticrew & Roberts 2005, Cruzes & Dybå 2011) of the relationship of product and portfolio management and agile software development.

For this, the following steps were taken; these are illustrated as well as further explained below.

1) Selecting a number of potentially relevant keywords
2) Organizing the keywords as keyword pairs
3) Searching Scopus[38] with all keyword pairs
4) Searching the Scopus database with five "extra" keyword combinations
5) Combining the hit lists from 3) and 4) and exclude multiple references to the same articles
6) Picking the potentially relevant scholarly articles based on titles (and abstracts, if necessary)
7) Combining the reference lists of all the potentially relevant articles
8) Picking the potentially relevant material (incl. Books) from the hit list resulting from step 7)
9) Combining the lists of potentially relevant articles from steps 7) and 8) and separating the potentially relevant books identified in step 8) into a list of their own
10) Adding 21 known-to-be relevant scholarly articles from outside of the searches' findings to the article list
11) Organizing the scholarly articles by topic and scoring them on relevance (scale 1-5)
12) Scoring the found books on relevance (scale 1-5)
13) Inserting the more relevant (3+) books into a "wish list" at Amazon.com to view recommendations, scoring the found books, and inserting those scoring 4-5 on relevance
14) Repeat step 13) until no new books are found
15) Removing less relevant books from the wish list one "relevance level" at a time; browse for new recommendations after each level; score found books and add to wish list
16) Repeat step 15) until the list consists of only of relevance level "5" books and no new recommendations scoring 4-5 for relevance are found
17) Searching Amazon using combinations of "lean", "agile", "software", "product management" and "portfolio management"
18) Scoring findings and inserting books of relevance level 4-5 into the wish list; repeat steps 15)-16)
19) Added two books that were known to be relevant but could not be found via the above steps
20) Extracting the relevant (level 4) scholarly articles and books (level 4+) for content
21) Assessing the conducted review for scholarly article coverage and content saturation

[38] SciVerse Scopus (or simply Scopus from hereon) is an abstract and citation database of peer-reviewed literature and web sources. For more information, see http://www.info.sciverse.com/scopus/about

These steps are illustrated in Figure 3.8 and they are explained in further detail below. Figure 3.8 also lists the keywords used.

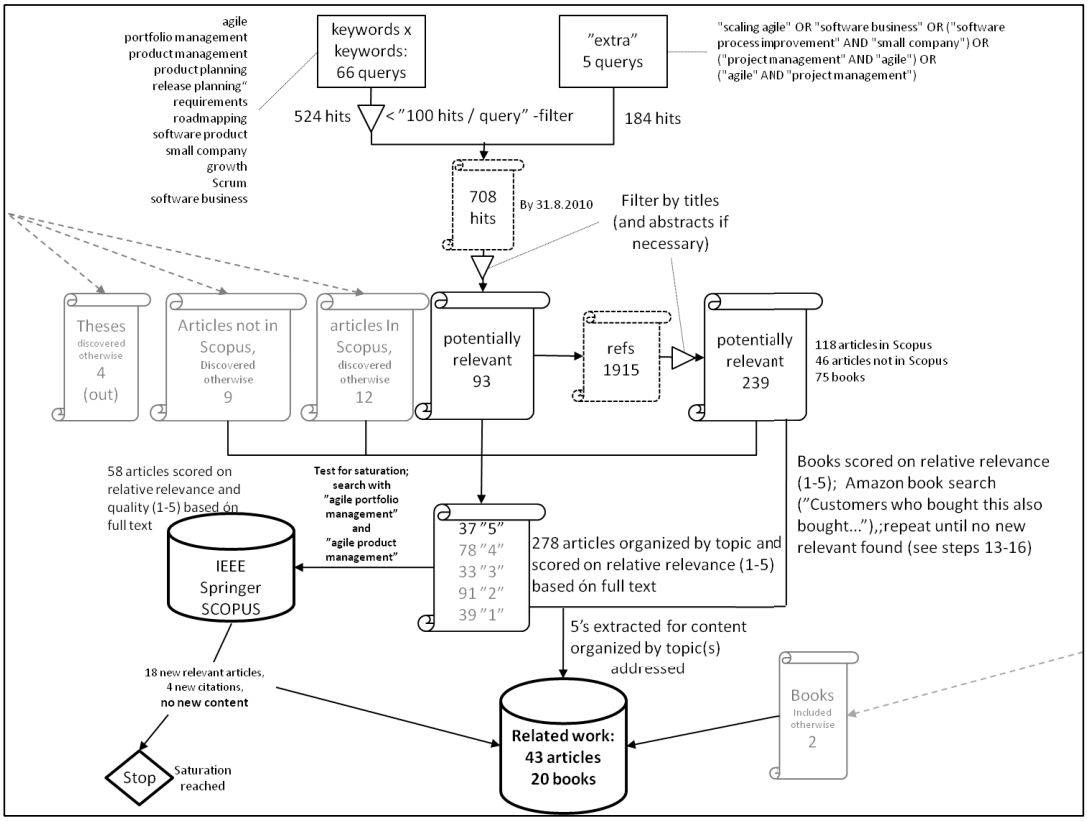


Figure 3.8 The steps conducted to answer RQ6

The paragraphs below explain the steps conducted in further detail.

Steps 1-2: First, we selected 12 potentially relevant keywords (see the top left of Figure 3.8) on the basis of the problem setting in Chapter 1 (*Introduction*). From these, we formed 66 *keyword pairs* because this study focuses on the combinations of, for example, agile software development AND portfolio management, instead of either alone. This also served to (in most cases) limit the amount of hits per search to a reasonable number.

Steps 3-4: We searched the Scopus database for all of the keyword pairs. Scopus was chosen over IEEExplore, SpringerLink and the ACM Digital Library, as Scopus includes both *engineering* as well as *management* articles (see section 2.1). In addition to searching with the keyword pairs, we also searched Scopus with five "extra" keyword combinations using the most relevant-seeming keywords (see the top right of Figure 3.8); all of the searches into Scopus discussed above were conducted on 31.8.2010.

Steps 5-6: After the keyword searches, we combined the hit lists from steps 3) and 4) to exclude multiple references to the same articles, yielding 708 distinct hits. From this list we picked the potentially relevant scholarly articles based on reading all of the titles, as well as abstracts, if there based on the title was any doubt about whether the article should be considered as potentially relevant or not. Of the 708 hits, 93 articles were seen as potentially relevant.

Steps 7-8: We combined the reference lists of all the potentially relevant articles, and removed multiple occurrences of the same material. This yielded a list of 1915 items for further examination, which we processed in a similar manner to the original 708 hits in step 6). However, as this list included, for example, book chapters, practitioner books, links to web sites as well, white papers by consultants, presentations, and even references to conversations, there was some differences in how different kinds of references were handled. Book chapters in scholarly books were treated as scholarly articles. In the case of practitioner books, we read those descriptions, and if necessary to decide on inclusion/exclusion, table of contents as well as book previews (for example, Amazon.com's "click to look inside" feature) and reviews that were readily available over the Internet. With the exception of columns by Steve Ambler in Dr. Dobb's Journal – as these were found directly via Scopus and classified as "scholarly material" – consultant white papers and other opinion-like materials were excluded, as were references to web sites, Scrum trainings as well as most presentations (all, except keynotes at scientific conferences, the contents of which were described in a paper of reasonable length). Picking the potentially relevant material from this list of references resulted in 164 new potentially relevant scholarly articles as well as 75 non-scholarly books. Of the scholarly articles, 118 could be found via Scopus while 46 could not.

Steps 9-10: The resulting list of 164 scholarly articles was then combined with the earlier list of 93 articles from step 6). In addition to these 257 articles, we decided to include 21 potentially relevant scholarly articles of which we were already aware of, but which were not found by the means described above. We were aware of these articles because they had been written by our colleagues, other people whose work we've been following, or they had recently been published in conferences or journals whose contents we had been following. Of these 21 additional articles, 9 had been published too recently to have been added to Scopus by 31.8.2010 when the searches were conducted. 12 could have in principle been found from Scopus with improving the searches. Thus, our list of potentially relevant material that needed further examination now comprised of 278 scholarly articles and 75 books.

Step 11: We organized the 278 scholarly articles by topic and subjectively scored on a scale from 1 (the worst) to 5 (the best) based on whether their contents were relevant for answering one (or more) of the research questions of this dissertation (see section 3.1). This was done based on skimming through the articles' full text. The number of articles for each of the "relevancy levels" was as follows: 37 (level 5), 78 (4), 33 (3), 91 (2) and 39 (1).

Step 12: We also scored the 75 non-scholarly books for their apparent relevance. This was done based on the material that was easily obtained online (see the description of step 8) above). The number of books on each of the "relevancy levels" was as follows: 1 (level 5), 8 (4), 11 (3), 39 (3) and 22 (1). Thus, of all the books reviewed, we could give the highest relevancy rating to only a single book[39] published in 2007. With the exception of one other discovered book published in 2009, all other books had been published before 2007.

However, a considerable number of the articles containing material relevant for this dissertation had been published between 2005 and 2010, and most of them were experience reports by practitioners published at conferences. Thus, we considered it likely that a significant amount of relevant material at least of the same quality than that already found might be missed if we did not specifically examine practitioner books published in 2007-2010 further.

Steps 13-14: Motivated by the possibility that our review might overlook a significant portion of related work, we set out to find more recent practitioner books and hoped to do this in a repeatable manner. For this, we decided to use the recommendations functionality of Amazon.com. We

[39] *Scaling Software Agility* by Dean Leffingwell (2007)

entered all of the books that had in step 12) received a relevancy score of 3 or higher into a "*Wish List*" at Amazon.com. Then, we used the "*Customers Who Bought Items in Your Wish List Also Bought*" functionality to browse for more potentially relevant books. New books that were discovered in this way were again scored from 1 to 5 in a similar manner as in step 12). Books that received a score of 4-5 and were not already in the possession of our research group were bought, as well as inserted into the Wish List at Amazon.com. This was repeated until no new books that would score 4-5 were found via the mentioned "*Customers Who Bought…*" functionality.

Steps 15-18: Then, the books that resided in our wish list but had a relevancy score of lower than 5 were removed one relevancy level (that is, first the "threes" and then the "fours") at a time, and the "*Customers Who Bought…*" functionality was used after the removal of each level to browse for new potentially relevant books. This yielded several additional books that were judged as relevant (that is, scored 4-5 for relevancy). After this, having obtained a degree of understanding on the keywords that were likely to discover relevant practitioner books from Amazon, we conducted some additional direct searches. In these, we used different combinations of the keywords "lean", "agile", "software", "product management" and "portfolio management". This yielded a final batch of six previously undiscovered books. One of these six books, one was scored as 5 for relevancy, and another as 4; the other four books were considered less relevant.

In total, the Amazon search in steps 13)-18) yielded a total of 43 potentially relevant new books. Of these, more than half were as relevant as the most relevant books discovered via the scholarly articles' references in step 8). Thus, in retrospect, we consider the conducted book search crucial from the perspective of conducting a successful review of the work related to product and portfolio management in lean/agile software development.

Step 19: Additionally, two books that were not found by the above described means were included in the material to be reviewed. The *PDMA Handbook of New Product Development* (Kahn, Castellion & Griffin 2005) was included because it contained useful definitions. Dean Leffingwell's book on *Agile Requirements* (Leffingwell 2011) was included because, although it was not yet published at the time we conducted the book search, we have been following the development of its contents in the Leffingwell's blog since Spring 2008. The book seems as one of the best sources of information currently available on matters related to agile product and portfolio management, and its inclusion saved us from having to refer to blog postings, white papers by consultants, companies, and so on – which should have been excluded by our search method anyhow[40].

Step 20: All of the books scoring 3-5 for relevancy based on the available online content were either already in the possession of our research group, or ordered in physical or electronic format (the latter preferred, if available). While waiting for the books to ship in, the 37 scholarly articles with the relevancy score of 5 were read through. The content relevant to answering one or more of the research questions was copied into a separate document, organized by topic, and iteratively re-organized as the need arose. After all the ordered books had arrived, we repeated this procedure for all books scoring 3 or higher for relevance. The final heading structure of sections 2.2-2.4 emerged this way.

Step 21: As a final step, we assessed whether our review had reached sufficient saturation content-wise, but also to double-check that we had not missed anything obvious. For this, we on 31.1.2011 searched Scopus, IEEExplore and SpringerLink. We chose to omit ACM Digital Library due to time and resource constraints. We used the following keyword combination:

[40] See steps 7) and 8)

"agile software development" AND ("product management" OR "portfolio management")[41]

This yielded 182 (Scopus), 106 (IEEExplore) and 53 (SpringerLink) hits, and after excluding duplicates, 58 articles were both undiscovered by previous searches, and based on their title, abstract and if needed, full text, judged as potentially relevant. Three of these articles were too new to have been discovered in our original searches into Scopus on 31.8.2010. Of the remaining 55 articles, six could not have been found from Scopus at all. Thus, we concluded that the relatively high number (49) of potentially relevant articles missed by the original searches was not due to our database selection, but because of the limitations of the original search query itself.

Instead of doing more searches and/or revising the entire database search, we took the approach of evaluating whether our review had reached content-wise saturation from the perspective of performing an adequate narrative review (Petticrew & Roberts 2005, Cruzes & Dybå 2011). For this, we skimmed through the texts of all of the 58 articles, and assessed them for relevance (1-5) as well as study quality (1-5; again, the higher the score, the better the quality). Six articles got a "5" and 12 a "4" for relevance, respectively. From the perspective of study quality, only one got a "5", three scored as "4", while the rest scored lower. However, we decided to process all of the articles in the same way as those articles that had received "5" for relevance in step 11).

In the end, out of the 55 newly discovered and not "too new" articles only four were cited: one by Petersen and Wohlin (2010) in section 1.1 (*Background*), one by Vanhanen et al. (2003) in section 2.2.13 (*Implications for work item management tool support*) and one in sections 2.3.5 (*Levels of portfolio management)* and 2.3.11(*Models for agile portfolio management*) each – these were by Steindl (2005) and Thomas and Baker (2008), respectively.

These four citations were significant only from the perspective of strengthening some of the conclusions that already been drawn based on the material found in earlier steps. Thus, from the perspective of the content-wise saturation of our narrative review (Petticrew & Roberts 2005), it seemed that even before the newly discovered scholarly articles, our review had already achieved sufficient coverage, possibly because of the book review. Thus, we decided to stop searching for new material.

The results corresponding to RQ6 are presented in chapter 2 (*Related work*).

[41] Because of the differing syntaxes of the search engines, the exact search query used was slightly different than the logical expression presented here

4 Results and discussion

In this chapter we summarize the contributions of this dissertation from the included publications, relate them to existing literature and answer the research questions (see also Figure 3.1 on p. 59). The limitations of the related studies are discussed before each research question is answered. The research problem (*How can product and portfolio management be linked with agile software development?*) itself is answered in chapter 5 (Conclusion) after a summary of the limitations of this study presented in this chapter.

Publications I and II outline and describe the key processes that connect top management and iterative & incremental software development. These are *product management* and *portfolio management*. Publication I presents a framework depicting an overview of these processes. Publication II presents a first outline of the key decisions involved. The contributions from publications I and II are summarized in section 4.1 (*Connecting business and software development*) below to answer RQ1 (*What are the key processes and decision areas that connect Business with Iterative and incremental software development?*).

Publication III discusses roadmapping of software products and the related services. It presents a model for visualizing product roadmaps and experiences from its use in three small software companies, along with some remarks regarding the state-of-practice of roadmapping based on the review of the related work in chapter 2. Based on these RQ2 is answered (*What does roadmapping mean in the context of agile software development and how* can a roadmap be visualized?) in section 4.2.

Publications IV and V discuss portfolio management. Publication IV describes why explicit portfolio management is important for at least certain kinds of small organizations that develop software. It summarizes problems that are known to arise from inadequate portfolio management, and exposes factors which dispose small software organizations that do not explicitly practice portfolio management to such problems. Based on these, RQ3 (*Do small software organizations suffer from the lack of explicit portfolio management?*) is answered in section 4.3.

Publication V presents a series of steps for setting up and conducting portfolio management developed in co-operation with three small software companies. These steps are combined with the portfolio management practices employed by a small software organization in publication IV. Based on these, we answer RQ4 (*How can portfolio management be set up and conducted in small software organizations striving for agile software development?*) in section 4.4.

Publication VI adds detail to the Business-to-Development connection presented in the other included publications by explaining how long-term plans and planning can be linked to daily tasks, and the role of portfolio management in moderating this link. Also, it presents Agilefant, a tool developed to support the described process. Complemented with the synthesis of the current literature on the subject as presented in chapter 2 (*Related work*), we answer RQ5 (*How can long-term product and business planning be linked to daily software development tasks in agile software development? What role does portfolio management play here?)* in section 4.5.

Finally, a note on the terminology: as this thesis addresses areas that have up to recent years been left outside of software engineering research, there was very little available in the form of an established terminology for long-term product and business planning when this work started. Thus, the terminology for describing and understanding the different artifacts, concepts and processes with respect to product management, roadmapping, release planning and portfolio management has, during the time the publications were written, largely been put together by the author from various sources. The terminology and concepts have also undergone several revisions both during and after

the period in which the publications included in this dissertation have been written. For the sake of readability and to better relate the contributions of the included publications to each other as well as the current literature, this summary part of the dissertation uses a uniform terminology that reflects the author's current understanding. Thus, while the terminology and some of the frameworks appear to differ slightly from those used in the individual included publications, the contents have not been changed. The "final definitions" of the key terms have also been summarized in Appendix A (*Glossary*).

4.1 Connecting business and software development

This section summarizes the contributions of this dissertation concerning the connection of business and software development processes and decision-making, with a particular focus on agile software development. This is done by conceptualizing *the connection between long-term product and business planning and iterative and incremental software development* (section 4.1.1, based on publication I) and identifying *the key decision areas that connect the perspectives of business and software development* (section 4.1.2, based on publication II). Section 4.1.3 discusses the limitations of the studies, and in section 4.1.4 we answer research question RQ1 taking the identified limitations into account.

4.1.1 An overview of the connection – the Cycles of Control

The study reported in publication I had the goal of conceptualising the connection between long-term business, product and release planning and iterative and incremental software development. This was approached by adding detail to the *Cycles of Control* framework (see Figure 4.1). The Cycles of Control (CoC) is a framework for describing iterative and incremental software development. It is based on the concept of time pacing, which refers to the idea of dividing a fixed time period allotted to the achievement of a goal into fixed-length segments (Takeuchi & Nonaka 1986, Gersick 1994, Eisenhardt & Brown 1998, MacCormack 2001). CoC has previously been described in e.g. (Rautiainen, Lassenius & Sulonen 2002, Rautiainen 2004, Rautiainen, Vuornos & Lassenius 2003, Rautiainen et al. 2002).

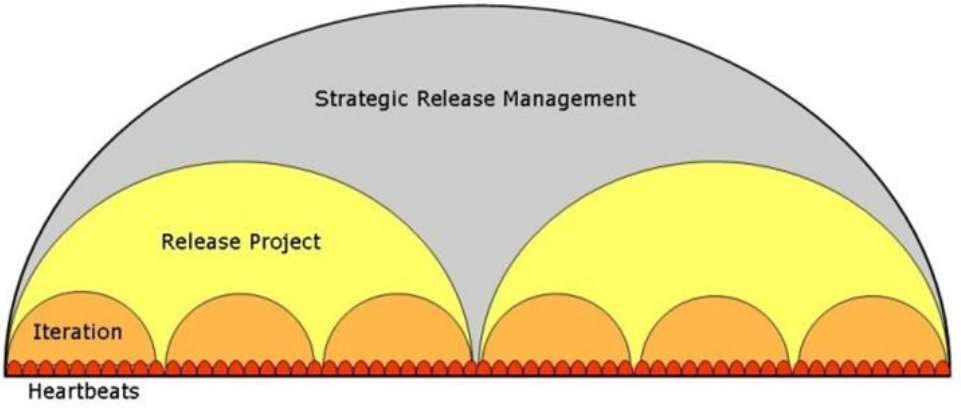


Figure 4.1 The Cycles of Control (Rautiainen 2004)

In Figure 4.1, the development of each individual release is managed as a time-boxed *project*. The projects are split into time-boxed *iterations*, which develop partial functionality for the product release. Daily work (Haapala 2010) is managed and synchronised in *heartbeats*, which represent the shortest time horizon; for example, the daily meetings of Scrum are an example of a heartbeat-level

practice. The longest time horizon depicted in Figure 1, *strategic release management*, represents the connection between Business and Software development decision-making.

(Vähäniitty 2004b, Vähäniitty 2004a) describe strategic release management as a cycle that has three main functions. First, it should connect top management with hands-on software development. Second, it is responsible for creating and updating the long-term product and release plans of the company. This includes decisions about the release cycles as well as the appropriate software development process. Third, in the strategic release management cycle, resourcing was considered from two perspectives: on one hand, it looks at identifying resource needs for the future and taking appropriate actions. On the other hand, the business perspective should participate to within-project decision-making for planning the individual development iterations. Also, as the development people or teams seldom have only a single product or task to attend to, the business perspective is also needed for managing the ongoing development activities as an explicit portfolio.

To more accurately describe the link between business and development, as embodied by the strategic release management cycle, we again applied the idea of time horizons and time pacing. This resulted in decomposing the strategic release management "cycle" into two cycles: *product management* and *development portfolio management*. Additionally, we explicitly included the link to top management as *enterprise* and *business management* cycles, respectively (see Table 7 below). Thus, in this summary, strategic release management refers to an area composed of the product and development portfolio management cycles (see Appendix A Glossary).

Table 7 Cycles of control in managing software development

Area	Cycles
Top management	Enterprise management, Business management
"Strategic release management"	Product management, Development portfolio management
Software development management	Release (project), Iteration, Heartbeat

The *enterprise management* cycle encompasses considering the overall direction of the company in terms of its various businesses, resource usage and general attitude towards growth. The *business management* cycle addresses the management of an individual business unit in terms of how to compete in a particular industry or product/market segment (Hitt, Ireland & Hoskisson 1997) and the business models employed (Rajala, Rossi & Tuunainen 2003). Together, these two cycles form the area called *top management*, and the set of decisions made are referred to as *enterprise*, and *business strategy*, respectively.

The *product management* cycle enacts the business strategy for a particular solution by creating the *product vision* (what is the essential value the solution will provide and for whom) and *release strategy.* Release strategy means setting a desired release cycle, stating where the solution should be going in terms of high-level functionality, changes to the underlying technologies and future resource needs. The release cycle often sets some basic requirements for the software development process in terms of development rhythm. Decisions made in this cycle are documented and communicated with the product vision and the roadmap.

The *development portfolio management*[42] cycle looks at the portfolio of current and immediately upcoming development activities as a whole and is concerned with making appropriate resourcing decisions in a timely and organised manner. The development portfolio management cycle ultimately connects business and development decision-making, and without a clear process for deciding which activities will be resourced and supported, decision-making tends to be reactive, causing shifting priorities, volatile resourcing and short-term fire fighting (Gill, Nelson & Spring 1996). Thus, effective development portfolio management is crucial for successful strategy implementation (Christensen & Raynor 2003).

As discussed in chapter 2 (*Related work*), current literature provided little advice in the way of understanding how the product management and development portfolio management cycles should be in practice be organised in the context of agile software development. In publication I we proposed that the presented cycles could be further characterised in terms of their *purpose*, *emphasis, forum*, *time horizon*, *process, interfaces* & *essential documents* and *key decisions*. *Purpose* refers to the fundamental reason the cycle exists. *Emphasis* describes what kinds of issues should be paid close attention to in the cycle. *Forum* refers to the roles chosen to represent the appropriate stakeholders on each time horizon. *Time horizon* refers to the length of the time perspective taken and specifies how far ahead this cycle considers the future in terms of planning and plans. The length of the time horizon for each cycle should ultimately depend on the rhythm of the market(s) involved, though some rules of thumb (e.g. iteration length should be 2-4 weeks) exist (Rautiainen, Lassenius, Vähäniitty, Itkonen, Mäntylä, Rusama & Vanhanen 2004). *Key decisions* refer to the "responsibilities" the cycle has in the overall picture of managing software development. *Process* refers to how and when the forum carries out its responsibilities. *Interfaces & essential documents* denote how the cycle in question links with the other cycles, in terms of input, feedback and output (e.g., documents, plans, decisions).

We also proposed that the *purpose*, *emphasis* and *key decisions* for each cycle stay similar for different companies. However, the implementation of the cycles (in terms of, e.g., forum, process, interfaces and essential documents) can and should vary.

Thus, as a partial answer to RQ1 (*What are the key processes and decision areas that connect Business with Iterative and incremental software development?*), business and development decision-making are connected by the *product* and *development portfolio management* processes involved. To fully answer RQ1, we undertook in publication II a more detailed examination of the key decisions these processes involve. The results are discussed in section 4.1.2 below.

4.1.2 Key decision areas connecting business and development

In publication II we, having in 2002 found very little guidance from the literature for helping small software businesses to link business and product development decision-making, took the approach of viewing strategic release management as the result of key decisions in managing software development. While the *how-to-succeed* varies over time both within and across companies and industries, *what issues are being decided on* remains fairly constant at a certain level of abstraction (Krishnan & Ulrich 2001). Thus, identifying these decisions and tailoring them to the context of software business would provide managers with a 'checklist' of what they should be paying

[42] We use the term *development portfolio* instead of a project or product development portfolio, because in many companies (especially small ones) there are activities that require a significant amount of attention from the developers, whether or not these activities actually are conducted as projects or involve product development in the strict sense. For details, see publications IV and V as well as the discussion of portfolio management in related work (section 2.3) in chapter 2

attention to.

The starting point for our work in compiling a framework of key decisions for strategic release management was a list of generic new product development decisions, or *decision areas* (Krishnan & Ulrich 2001). The list comprised of two categories: decisions in setting up a development project (*product strategy and planning*, *product development organisation* and *project management*), and decisions within a project (*concept development*, *supply chain design*, *product design*, *performance testing and validation* and *production ramp-up and testing*).

We tailored these decision areas to comply with the characteristics of small companies and the software business as presented in the literature, for example (Condon 2002, Fayad, Laitinen & Ward 2000, Laitinen, Fayad & Ward 2000, Ward, Laitinen & Fayad 2000, Regnell, Beremark & Eklundh 2000). The validity of our initial 'key decision areas' for the small software product business context was evaluated by whether examining three small software companies would be meaningful using such a structure, and whether relevant problems, challenges and improvement suggestions could be found using the decision areas as a checklist. As the result of these case studies, decision areas (or parts thereof) were added, removed, renamed and restructured to better cover the issues of importance that surfaced during the case studies. For details, see publication II and Vähäniitty's master's thesis (2003).

The resulting key decisions have been grouped into the decision areas of *portfolio management, organisation*, *development model*, *product management* and *quality strategy*. Although all of the decision areas may not be equally topical or strategic to a company at a given time, a company that recognises the scope of the decisions involved can *explicitly* decide not to address certain issues at a given time, as opposed to not even being aware that such issues could be important.

4.1.3 Limitations

The most important identified limitations of the research methods used in order to answer RQ1 are highlighted with a red dashed outline in Figure 4.2 and discussed below in chronological order.

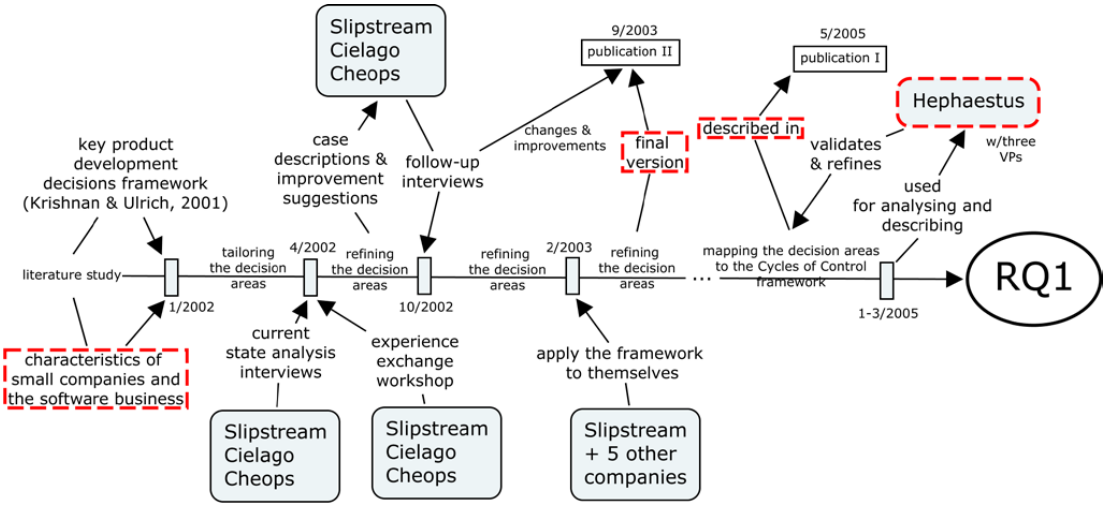


Figure 4.2 The most important identified limitations of the methods used for answering RQ1

The literature review of the characteristics of small companies and the software business conducted for publication II was not systematic. Thus, it is likely to have overlooked some important sources.

However, from the perspective of answering RQ1, this shortcoming is probably resolved due to the amount of iterative revising of the key decisions framework based on the industrial collaboration with multiple small software companies.

As for the key decisions framework itself, it can hardly be said that any of the individual key decision areas would be specific to small companies. While the naming, structuring and contents of the framework are largely a result of the empirical context, it may also be suitable to software development in general. Still, we have not examined the scalability of the key decisions framework to larger companies and make no claims towards this.

The mapping of the portfolio management and product management key decision areas to the Cycles of Control presented in publication I was validated by analysing Hephaestus. As Hephaestus was a medium-sized company, the choice of the case company may have helped in generalizing on the results. While the analysis and the discussions that took place reflect on the current size, processes and challenges at Hephaestus, all of the involved vice presidents were working for Hephaestus when it still was a small company. When explicitly inquired of the matter, the interviewees were of the opinion that the framework from publication I would be generic enough to be used for analysing the Business-Development decision-making at both smaller as well as larger companies. Thus, while we still make no claims towards the scalability of the framework, we view the selection of the case company adequate for the purposes of answering RQ1.

The description of the expansion of the Cycles of Control framework in publication I lacks, due to the space limitations, the verbosity needed to describe all of the subtleties that originated during the interviews when validating the framework. From the perspective of answering the research problem, this limitation is addressed in publication VI, which adds more detail to the framework, as well as the longer descriptions of the framework and the respective tool support in section 4.5 (*Linking long-term planning and daily work*).

4.1.4 Answering research question #1

Having summarized the results from publications I and II as well as the respective limitations, we are now ready to answer RQ1:

What are the key processes and decision areas that connect Business with Iterative and incremental software development?

Business and Development decision-making are connected by *strategic release management*, whose key decision areas are *portfolio management*, *organization*, *development model*, *product management* and *quality strategy*. From these areas, *roadmapping* (as part of product management) and *development portfolio management* (that is, the short-term resourcing aspects of portfolio management) play the most crucial part in connecting Business and Development decision-making.

Based on the review of the related work in chapter 2, development portfolio management is addressed very little in existing literature – especially when it comes to agile software development – and roadmapping lacks a clear example that fits the small software organization context. Furthermore, the nature roadmapping is not well understood in the context of agile development processes. Based on the case studies conducted in publication II as well as the related work, practitioners also seem to view these areas as the most challenging.

The sections that follow summarize the results of publications III (section 4.2), IV (section 4.3) and V (section 4.4) to shed further light on roadmapping and development portfolio management, respectively.

4.2 Roadmapping

This section summarizes the contribution from publication III concerning the roadmapping of software development efforts. These are *a model for visualizing product roadmaps* (section 4.2.1) and some further observations to *the state-of-practice of roadmapping* based on the three case companies (section 4.2.2). Section 4.2.3 discusses the limitations of the study, and in section 4.2.4 we answer research question RQ2 taking the identified limitations into account.

4.2.1 A model for visualizing roadmaps

The following is a description of the roadmap visualization adapted from an article by Wells et al. (2004) and further developed together with the case company ToolCo for publication III. The terminology and concepts used in publication III have below been adapted from to better match contemporary literature on agile software development (see section 2.2).

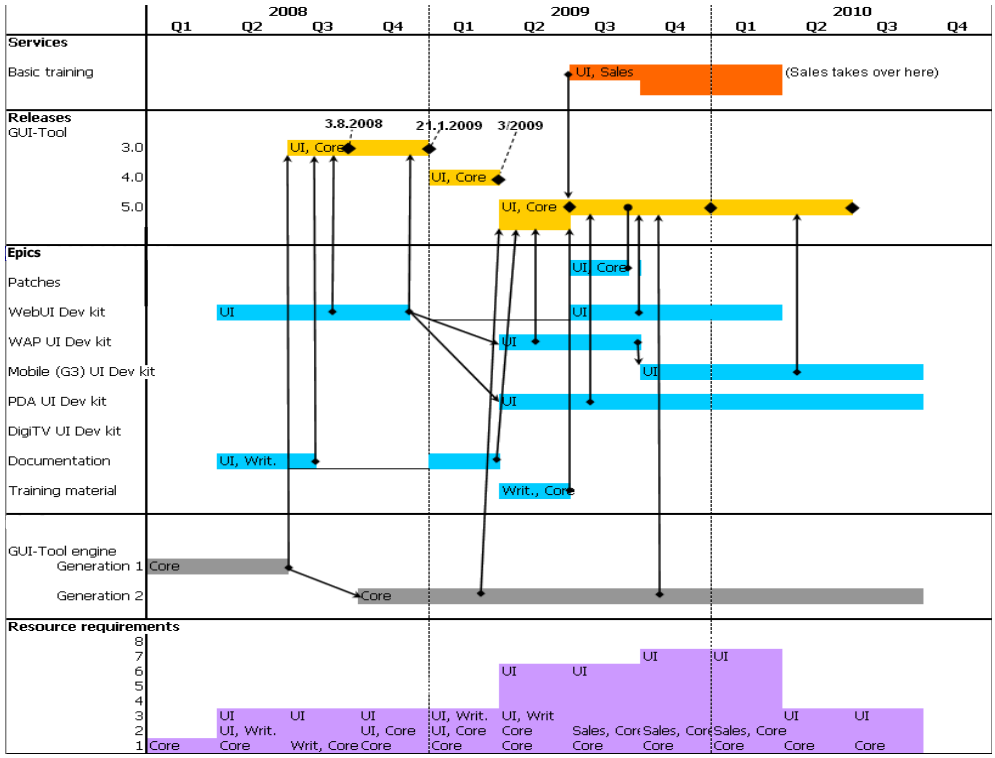


Figure 4.3 The model for visualizing product roadmaps from publication III

The roadmap (Figure 4.3) expresses the services that require attention from the development people (orange top layer), release and development schedules for the product(s) (yellow layer; diamonds denote releases), the Epics that are planned to be worked on (blue layer; arrows denote internal releases), and the amount of effort that the activities in the roadmap are currently estimated to require (purple bottom layer; on each of the layers, the height of a vertical bar denotes the estimated amount of effort needed). Together with one of the case companies, ToolCo, a four-step process for creating and updating product roadmaps was created. See publication III for a more detailed explanation of the semantics of the roadmap visualization.

4.2.2 Observations regarding roadmapping state-of-practice

Developing, applying and discussing the model for visualizing product roadmaps in co-operation with three small software companies, ToolCo, TeamCo and MobAppsCo resulted in several observations that may reflect the state-of-practice of roadmapping in similar companies.

We noticed a common conceptual view of the product and its Epics and Features may be lacking even when the development organization is small. Second, even in small companies, possible long-term plans, strategic ambitions and business implications that the top management has envisioned may not be visible to the rest of the organization if attention is not explicitly devoted to communicating these, for example, by using hierarchical work item structures (see sections 2.2.10-2.2.13). Third, the servicing needs of already launched products may be considerable, and a failure to account for them jeopardizes the sustainability of the development efforts.

Finally, in all of the case companies it seemed that preparing and updating roadmaps as separate, detailed documents tends to lead into them not being updated. This observation is also supported by Lehtola et al. (2005) as well as much of the recent related work on agile software development as discussed in section 2.2. As an example of this from publication III, MobAppsCo's management had in the past conducted roadmapping by writing a document that described as closely as possible the platform, the set of applications and their features as a function of time. However, the approach felt too cumbersome, and the document was not kept up-to-date. The practice was scaled down to having one or two bulleted pages with basically the same information but with less detail and a shorter time range. The interviewee at MobAppsCo considered it feasible to apply the visualization for co-ordinating the more traditional R&D-type work with prioritising, selecting and planning customer-initiated development projects. The visualization could then help in communicating the schedule and resource implications better.

Thus, in hindsight, this author presumes that the value of the roadmap visualization from the practitioner perspective may lie first and foremost in providing a visual checklist of the issues and aspects that should be considered as part of long term planning. On the other hand, with proper tool support for backlog management, a visualizing the roadmap view into the product backlog (see section 2.2.12) could be generated automatically. We will return to discuss this further in section 4.5.

4.2.3 Limitations

The most important identified limitations of the research methods used in order to answer RQ2 are highlighted with a red dashed outline in Figure 4.4 and discussed below.

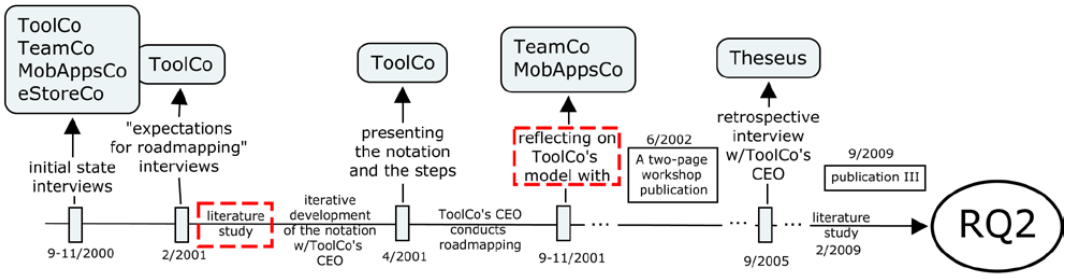


Figure 4.4 The most important identified limitations of the methods used for answering RQ2

The literature review conducted to find examples and information on roadmapping for the basis of developing an initial version of the notation for ToolCo was not systematic. Thus, it has possibly

overlooked some important sources. However, this shortcoming is alleviated by two issues: First, in addition to the author, three other people were also conducting the literature review, possibly increasing the resulting coverage. Second, we conducted another (again, non systematic, but performed together with a colleague) literature study during the preparation of the work for publication in the beginning of 2009; see section 3.3.2 (*Approach and methods for* answering RQ2) for details.

The second limitation of possible importance is the fact that neither the notation nor the steps for conducting roadmapping have not, to the knowledge of the author been applied in companies other than ToolCo. Thus, while the elements depicted by the visualization were agreed by all of the case companies to be of importance in conducting roadmapping, we make no claims towards whether companies should use exactly this kind of visualization or not. On the contrary, it is quite likely that at least in the context of agile software development, a complex roadmap that is separate from the product backlog itself is quite likely to become quickly outdated (Vidgen & Wang 2009, Wilby 2009, Hodgkins & Hohmann 2007).

As formulated by RQ2, the intent of including publication III is to provide a tangible example of how a small company actually conducted roadmapping, as well to provide basis for defining the concept of roadmapping and roadmaps in a way that is compatible with how long-term planning should be understood in agile software development (see section 2.2.12). Thus, the author considers that the above mentioned shortcomings to have limited impact in terms of answering RQ2.

4.2.4 Answering research question #2

Having summarized the results from publication III and discussed the identified limitations we are now ready to answer RQ2 (*What does roadmapping mean in the context of agile software development and how can a roadmap be visualized?*).

Drawing from publication III and the related work discussed in section 2.2 (*Product management*), we answer RQ2 by providing the following, agile compatible definitions of 'roadmapping', 'roadmap' and 'release planning':

> **Roadmap** is a view into the product backlog that depicts how a particular solution (or line of solutions) is currently planned to evolve in terms of high level goals in the foreseeable future. The roadmap shows the release dates, the planned features (possibly with related Epics and investment themes) and their estimated sizes in story points. The roadmap should also depict accompanying services, and the planned resource usage for those services that demand the developers' attention. Ideally, a visual roadmap is possible to be discerned directly from viewing the product backlog itself, or can be automatically generated.
>
> **Roadmapping** refers to grooming the product backlog so that the high level goals for the foreseeable future are visible and reflect the current understanding of the relevant stakeholders. In contrast, **release planning** refers to the planning and continuous refining of the contents of the immediately upcoming release.

An example notation for visualizing roadmaps is presented in Figure 4.3. It expresses the release and development schedules for the product(s), the Epics that are being worked on, the services that require attention from the development people, and the resources that the activities in the roadmap are currently estimated to require.

4.3 Do small software organizations need portfolio management?

Here, we summarize the results from publication IV in order to answer RQ3 (*Do small software*

organizations suffer from the lack of explicit portfolio management?). Section 4.3.1 describes the problems associated with inadequate portfolio management in the literature, and in section 4.3.2 these are compared to the problems experienced by the case companies. Section 4.3.3 discusses the limitations of the study, and in section 4.3.4 we answer research question RQ3 taking the identified limitations into account.

4.3.1 Problems associated with inadequate portfolio management

Portfolio management decisions always get made – sometimes consciously, but also inadvertently, through inaction, or by accident. Thus, the lack of an explicit portfolio management process does not necessarily cause problems: the mix of ongoing activities in a small organization may be simple enough to be managed project-wise, or even without formal project management. For example, if the ongoing activities have no resource or deliverable dependencies, explicit portfolio management may not be needed.

To assess whether an organization is actually suffering from the lack of explicit portfolio management, we in publication IV set out to find what symptoms occur in conjunction with inadequate portfolio management. If an organization exhibits many or most of such symptoms, but does not consciously practice portfolio management, it is reasonable to propose that explicit portfolio management could be beneficial. The following problems were identified from existing literature to be symptomatic of inadequate portfolio management: 1) *Excessive multitasking*, 2) *Firefighting*, 3) *Overload*, 4) *Ineffective decision-making*, 5) *Missing strategic alignment*, 6) *Slipping schedules*, 7) *Project failures and poor profitability* and 8) *Perceived need to improve project management*. These are further described in publication IV.

4.3.2 Problems experienced by the case organizations

In publication IV the problems listed in section 4.3.1 were contrasted with the problems experienced by five case organizations that did not practice explicit portfolio management, as well as one case organization that did. Based on the comparison, we concluded that it explicit portfolio management seems relevant for small software organizations at least when certain conditions hold.

Our case organizations had four common denominators we suspect may predispose similar organizations without explicit portfolio management to problems. These were 1) *leveraging customer-specific projects for product development*; 2) *dealing with a portfolio of different kinds of activities instead of a clear-cut product, project or product development portfolio*; 3) *the developers' multiple and sometimes conflicting roles and responsibilities*; and 4) *recent growth*.

4.3.3 Limitations

The most important identified limitations of the research methods used for answering RQ3 are highlighted with a red dashed outline in Figure 4.5 and discussed below.

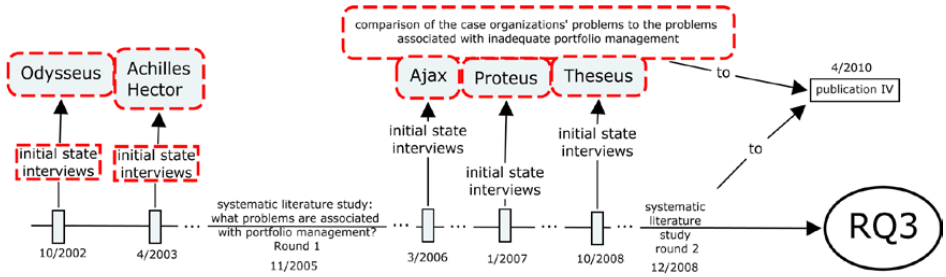


Figure 4.5 The most important identified limitations of the methods used for answering RQ3

The most crucial limitation for answering RQ3 is the possibility of a biased selection of case organizations due to our existing co-operative relationship. The case organizations are typical when compared to all of the case companies discussed in the included publications (see Table 5 on page 61). However, the author does not have evidence beyond anecdotes of how typical our cases are of the entire population of small software organizations. It can also be questioned whether organizations who – like the case companies examined in this dissertation – experience pressures in their long-term planning and portfolio management practices are an actual majority in the population of small software organizations. For now, these factors are accounted for in the answer to RQ3 in section 4.3.4 below, but they should be further examined in the future.

Because the data for Achilles, Hector, and Odysseus was not originally gathered to study portfolio management per se, our analysis had to rely on less detailed data for these cases. Thus, the comparison was made for only those problems that were experienced by all of the case organizations. Second, the categorization of the symptoms of inadequate portfolio management can affect the results of the comparison. To control for this bias, we matched the case organizations' problems also against two symptom lists found in the literature by Elonen and Artto (2003) and Blichfeldt and Eskerod (2008). The results of these 'control comparisons' were in line with the results of the comparison to our synthesised problem list. Thus, it is plausible that our categorization of the problems does not cause significant bias for the purposes of answering RQ3.

We conducted our literature study on problems associated with inadequate portfolio management after the data at Achilles, Hector, and Odysseus had been collected. The bias of looking for evidence in support of the hypothesis – in other words, that the organizations would in fact be suffering from inadequate portfolio management – is difficult to avoid (Patton 2002). In this case, the fieldwork may have affected the literature study. To limit the potential bias, we attempted to conduct our literature review in a systematic and reproducible manner, using a defined protocol to, for example, include and exclude material. However, we did not systematically examine the reference lists of the articles produced by the database search. Nevertheless, the end results seem adequate for the intended comparison. Striving for a systematic approach with a defined protocol in the literature review can also be considered an improvement when compared to many contemporary software engineering studies (Kitchenham et al. 2009, Cruzes & Dybå 2011).

Finally, the author acknowledges that the problems experienced in the case organizations are likely to have root causes in addition to, or aside from inadequate portfolio management. However, the perceived improvements at Theseus, the single organization that had taken strides toward explicit portfolio management are encouraging. While the underlying set of problems can be complex, explicit portfolio management may still be a reasonably effective remedy.

4.3.4 Answering research question #3

Based on publication IV we answer RQ3 (*Do small software organizations suffer from the lack of explicit portfolio management?*) as follows:

Small software organizations seem to suffer from the lack of explicit portfolio management at least when the following conditions hold:

1) The organization is leveraging customer-specific projects for product development,
2) the development personnel possess multiple roles and responsibilities and are concurrently performing many different types of activities
3) the organization has recently grown, or the management intends to grow the organization in the near future

Recent literature regarding agile software development also seems to be in support of these conclusions (see section 2.3.5-2.3.11 for details).

4.4 Portfolio management for small software organizations

In this section we summarize results from publications IV and V in order to answer RQ4 (*How can portfolio management be set up and conducted in small software organizations striving for agile software development?*). In the first two sections we summarize our approach for setting up portfolio management (section 4.4.1), and describe the practices involved (section 4.4.2). Section 4.4.3 discusses the limitations of the study, and in section 4.4.4 we answer research question RQ4 taking the identified limitations into account.

4.4.1 Setting up portfolio management

Publication V contains a four-step approach for setting up portfolio management. First, the activity types that comprise the development portfolio are identified. Then targeted spending levels are set. Third, suitable ways to manage the different types of development activities are identified in terms of development cadence and the iteration control points are synchronised where possible, thus forming the basis for the portfolio management process. Last, the portfolio management process itself is set up through defining the control points to govern the portfolio.

4.4.2 Portfolio management practices

In addition to the practices described in publication V, publication IV also describes a set of practices that Theseus, a small software development organization, used to alleviate problems stemming from inadequate portfolio management. Theseus refrained from assigning developers to more than two concurrent projects, kept track of who was assigned to what using an internally developed toolset; tracked the developers' workloads using the toolset, and accounted for projects' realized billing in sales peoples' incentive systems (instead of sales achieved).

Section 4.3.4 below contains a synthesised list of portfolio management practices based on both publications V and IV.

4.4.3 Limitations

The most important identified limitations of the research methods used in order to answer RQ4 are highlighted in Figure 4.6 and discussed below.

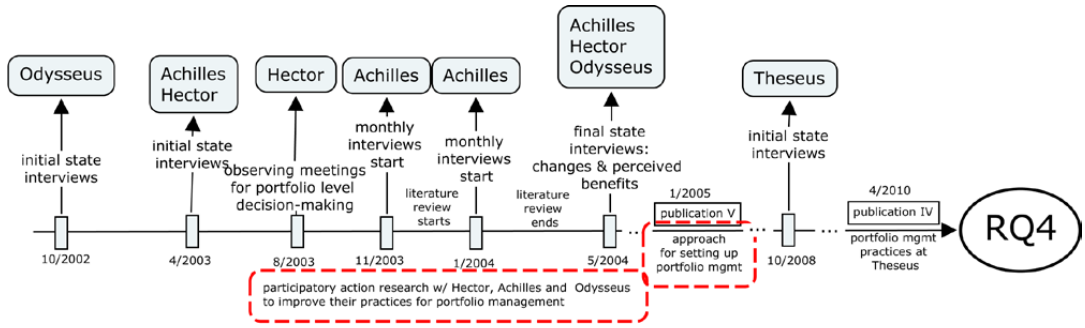


Figure 4.6 The most important identified limitations of the methods used for answering RQ4

First, the reported usefulness of our approach to setting up portfolio management is based on the perceptions of one person at each case company (the interviewee) and the interpretations of the researchers, not on explicit measures of success or failure. Second, the observation period is too short for evaluating how lasting the changes are.

Because we employed participative action research, researcher bias is a factor to be considered. Both the authors and the interviewed product development managers have a vested interest in producing good results, and negative evidence may have been accidentally overlooked. Also, is difficult to evaluate the value of the presented approach as such, eliminating the contribution of the authors – for this, a longer observation period and more cases, combined with a different research setting are needed. Nevertheless, to provide access for further validation of the findings, we have during the study kept records of all discussions, archived all relevant documents and taped all interviews.

Overall, it seems obvious that progress was made in each company. Each company did set up an explicit portfolio management process, which was both being used and refined. Additionally, portfolio management as it was practiced at Theseus included most of the aspects prescribed by our approach from publication V. This is also supporting evidence of the practical utility of portfolio management in those small software organizations that are similar to the case companies described in publications IV and V. Due to the unknown representativeness of the sample, generalization of the results beyond companies similar to those discussed in publications IV and V cannot at this point be made.

4.4.4 Answering research question #4

Based on the results from publications IV and V, RQ4 (*How can portfolio management be set up and conducted in small software organizations striving for agile software development?*) is answered as follows:

The key steps in setting up and performing development portfolio management in small software organizations similar to the case organizations described in publications IV and V are:

1) naming a group of people to be responsible for portfolio-level decision-making
2) building a publicly visible list of all ongoing activities that require time from development, including the information on who are assigned to which activity
3) synchronizing the portfolio
4) meeting regularly at portfolio synch-points (for example, on a bi-weekly basis) to keep the list of ongoing activities up-to-date, perform short-term prioritization (force-ranking the ongoing activities) and setting the default resource allocation until the next meeting

5) agreeing on how decisions affecting more than one ongoing activity are made in urgent, 'emergency' type situations
6) identifying the different types of development activities
7) setting target spending levels per development activity type that reflect the organization's strategy, and possibly tracking the actual spending
8) curbing excessive multi-tasking by explicitly setting limits to the number of concurrent activities a person can be involved in
9) tracking the developers' (or development teams') workload
10) ensuring that incentive systems do not steer people towards local optimization

4.5 Linking long-term planning and daily work

In this section we summarize results from publication VI in order to answer RQ5 (*How can long-term product and business planning be linked to daily software development tasks in agile software development? What role does portfolio management play here?*). In section 4.5.1 we describe our framework for linking long-term product and business planning and portfolio management with development teams' daily work. Section 4.5.2 describes Agilefant, the proof-of-concept support tool for evaluating and further refining our framework. Section 4.5.3 discusses the limitations of the study, and in section 4.5.4 we answer research question RQ5 taking the identified limitations into account.

Because of the page limits of publication VI itself, as well as the fact that this section leverages much of the results from the other five publications as well as the review of related work, we have reserved more space for this section than any of the previous four sections that summarize the results from publications I-V.

4.5.1 The framework

Publication VI describes our framework for linking product management – or, more specifically, the roadmapping and release planning processes – with agile software development. The essentials of the framework are illustrated in Figure 4.8, Figure 4.7 and Figure 4.9 below. The terminology and concepts have been adapted from publication VI to better match the recent developments in the literature on agile software development (see section 2.2.10) as well as the author's current understanding.

Figure 4.7 depicts how the enterprise strategy sets constraints for business level goals, which are linked with the developers' daily work via a tree structure of hierarchical goals. These "goals" are commonly referred to as simply work or backlog items. Goals possible to achieve in a single iteration or release are referred to as *Stories* and *Features*, respectively. Goals that are considered too big to fit into a single release are referred to as *Epics*. Epics should make the achievement of explicit *Business goals* possible.

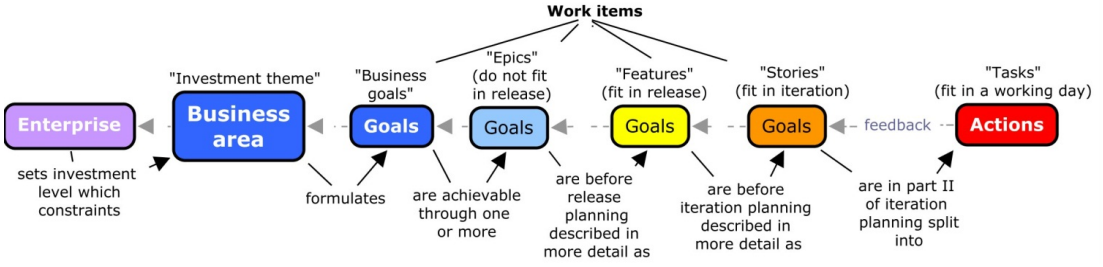


Figure 4.7 From enterprise strategy to goals, actions – and back again

Figure 4.8 below relates the work item hierarchy depicted in Figure 4.7 to the sections of the product backlog (2.2.6) and levels of planning in agile software development (see section 2.2.7).

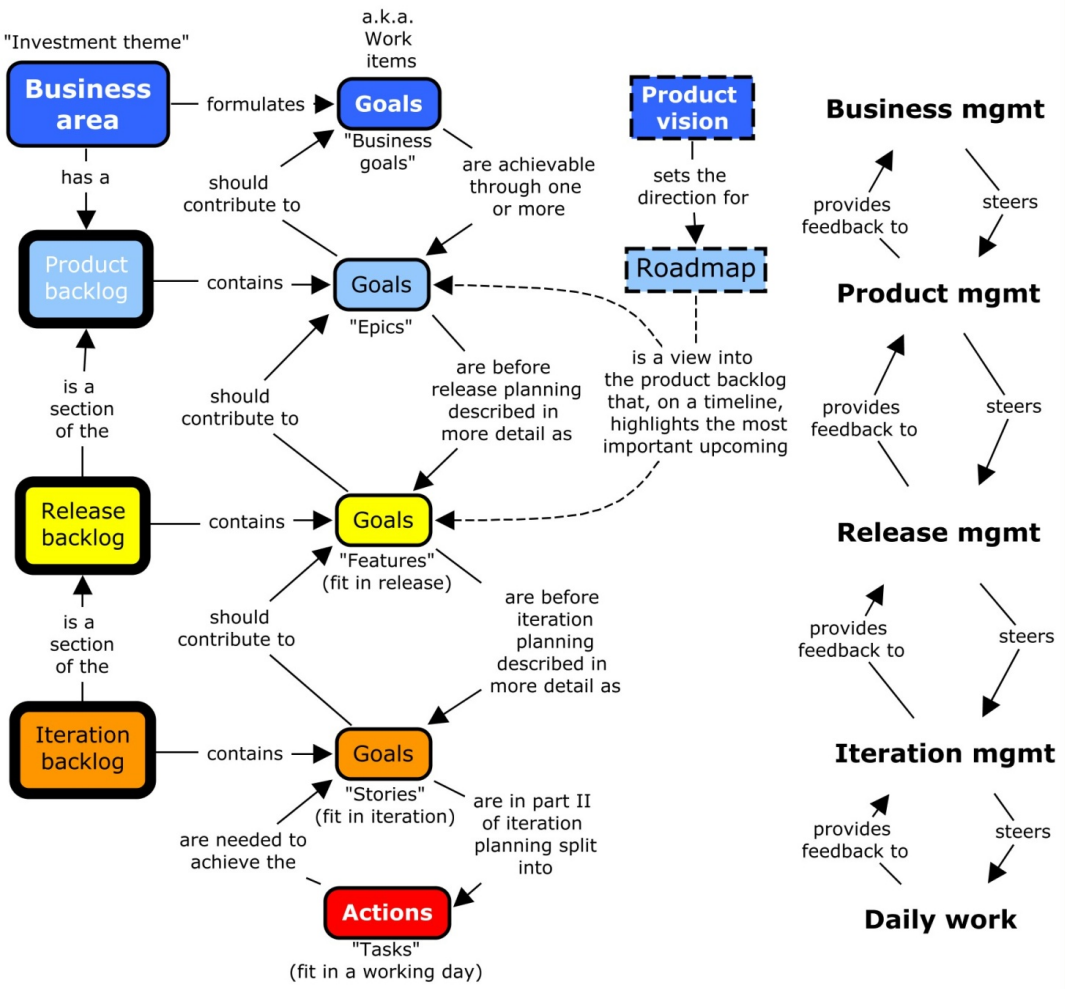


Figure 4.8 The product backlog, planning levels and work item hierarchy

In line with the definitions of roadmapping and release planning that were used to answer research question RQ2 in section 4.2.4, the *Roadmap* in Figure 4.8 is a view into the product backlog that highlights the most important future Features and Epics that are planned to be pursued in the foreseeable future. The product vision in turn sets the long-term direction for the product. The Product vision and Roadmap concepts are denoted with the dashed line in Figure 4.8 (Product vision and Roadmap), meaning that have not at the time of writing this been implemented in Agilefant as explicit objects of their own; this will be further discussed in section 4.5.2 (*Tool support*).

To complement the single business area view presented in Figure 4.7 and Figure 4.8 above, Figure 4.9 below illustrates our framework from the perspective of portfolio management.

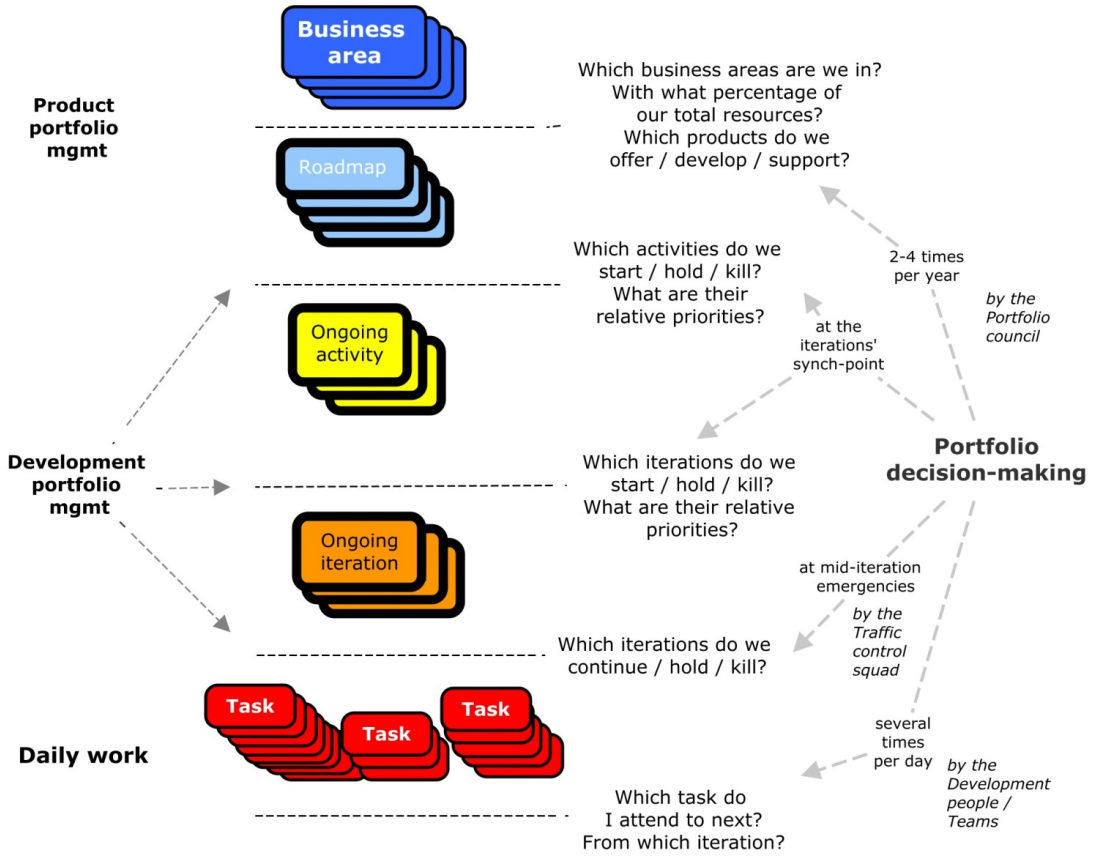


Figure 4.9 Portfolio management moderates the flow of strategy-to-action

Portfolio management decision-making on different levels moderates the flow from strategy to action and back in agile software development (see sections 2.3.5-2.3.11 and Figure 2.11). Instead of "releases" or "projects", Figure 4.9 uses the generic term *activity* to highlight that everything that takes up time and attention from the development teams or people should be included in the development portfolio: for example customer-specific development, consulting, possible non-project work and so on[43]. In many of the case companies studied for the included publications, especially 'non-project' activities seemed to take up an amount of effort from development people's that upon a closer look was both considerable, as well as a surprise for both the developers and the managers involved.

Figure 4.9 also names the parties responsible for making the decisions on each level. These are further described below. We have chosen to name[44] the three parties responsible for decision-making on the different levels depicted in Figure 4.9 as the *Portfolio council* and the *Traffic control squad*, and the *Developers* (or cross-functional *Teams* in the context of "by-the-book" agile

[43] See the notion of *types of development activity* in publications IV and V, and *project types* in publication VI.

[44] The terms *Portfolio council* and *Traffic control squad* have been adopted from case companies *Janus* and *Hector*, respectively (see Table 5 on p. 61)

software development). The *portfolio council* is composed of the people responsible for the success of the enterprise, and includes representation from development. The portfolio council is responsible for *product portfolio management* and ultimately, *development portfolio management* decisions as well. Both of these are further described below. The *traffic control squad* is responsible for resolving *mid-iteration conflicts and crises*. In other words, while it does not launch or kill activities, it does manage the 'traffic' in the product development pipeline. The *traffic control squad* is composed of a subset of the portfolio council as well as representatives from development on a per-need basis for resolving the conflict in question.

Product portfolio management refers to deciding on the set of products and services offered and developed by the organization, as well as deciding on the relative spending across the business areas. This roughly corresponds to the notion of deciding on investment themes and the percentage of total resources of the enterprise to spend on which theme (see section 2.3.6 *Portfolio management as investment level setting*). In contrast, *development portfolio management* deals with tactical resource allocation and prioritization across the set of possible activities that compete for the same pool of resources. While product management is responsible for prioritizing and preparing the backlog of a certain activity (see Figure 4.8), development portfolio management decides according to the situation at hand which of the activities get attended to, and what their relative priorities currently are.

In line with publication V, our framework advocates synchronizing the planning and decision points for all of the ongoing activities that require attention from the development people. For a portfolio of development activities that adhere to a "pure agile" life cycle, this means synchronizing the sprints of all those ongoing activities that utilize the same resource pool, even if all of the activities would not be concerned with the same business area. In these synch-points, the portfolio council evaluates, selects and prioritizes new activities, prioritizes ongoing activities, and allocates resources based on business priorities and other constraints such as resource or technical dependencies. Thus, the synch-points correspond to the notion of portfolio reviews as discussed by for example Cooper et al. (2002). The notion of sprint synchronization is also supported by virtually all of the discovered practitioner literature that discusses portfolio management in the context of agile software development (see sections 2.3.8 and 2.3.11). Synchronizing the sprints makes it feasible to commit the resources for a fixed period. Provided that the organization's resulting 'joint sprint' is short enough, it is possible that this helps alleviate a firefighting mentality as cross-project trade-offs are possible to make proactively and on a more continuous basis.

Ideally, the portfolio council is able to set and keep the resource allocation fixed for the duration of the organization's joint sprint. However, business realities may make an absolute adherence to this principle next to impossible. This is especially the case in those organizations where the development portfolio does not consist solely of activities following a "pure agile" life cycle. As this was the case in all of the case organizations studied in the publications included in this thesis, it is plausible that such a situation can be fairly common. In our approach, the fact that mid-sprint conflicts eventually occur is addressed by appointing the *traffic control squad*, a forum in which mid-iteration conflicts between activities can be resolved in the light of the business priorities. The traffic control squad is a subset of the portfolio council, consisting of only those people necessary to solve the conflict in question, possibly terminating or freezing one or more ongoing iterations so that the most important ones get the needed support. Based on the experiences from Theseus – the small organization in publication IV that did explicitly practice portfolio management – having a nominated traffic control squad may increase the chances for systematic and conscious portfolio decision-making to take place when mid-increment conflicts occur.

In our framework, portfolio decision-making is also considered to occur during daily work, as the development people choose which task(s) they will next attend to. This decision-making is trivial in

the case of "pure agile", since a development team (or developer) has only a single sprint backlog to pull work items from. However, as seen in the included publications, it may well be that this is seldom the case in real-world organizations, no matter how small. Thus, the development teams and developers may have several different activities that compete for their attention. And, even in the case of all the competing activities having up-to-date and prioritized backlogs, it may not be possible to in the mathematical sense determine what the 'next most important task' to attend to is.

For example, which is more important: to attend to the tasks needed to accomplish story #4 of the most important activity, or the tasks needed to accomplish story #1 of the activity that has been ranked as second in importance for the ongoing set of iterations? Thus, while these kinds of decisions can be escalated to the traffic control squad and/or the portfolio council, portfolio decision-making on the level of daily work always relies somewhat on the developers' intuition and ability to judge if asking for help in prioritization is necessary. While supporting portfolio decision-making on the level of daily work is somewhat out of the scope of this dissertation, it will be briefly touched upon in section 4.5.2 below from the perspective of tool support to highlight its importance.

4.5.2 Tool support

The missing conceptual links between agile software development and product and portfolio management are reflected in project management/issue tracking tool support as well (Goth 2009, Treude & Storey 2010). As an example, the solutions for project management and issue tracking we have up to until quite recently[45] seen used in the Finnish software industry either 1) have poor support for agile software development 2) lack the capabilities to link daily work such as tasks and user stories with long-term product and business objectives, and/or 3) do not support managing the developers' efforts as an explicit portfolio.

Agilefant is a proof-of-concept prototype support tool for backlog and development portfolio management as depicted in Figure 4.7 (*The product backlog, planning levels and work item hierarchy*), Figure 4.8 (*The product backlog, planning levels and work item hierarchy*) and Figure 4.9 (*Portfolio management moderates the flow of strategy-to-action*). It is an open source J2EE web application based on the Spring framework[46]. Agilefant's development has between 2006 and 2008 been largely directed by the author of this dissertation, with a larger team joining in and collaborating from the beginning of 2008 onwards[47]. Since its inception, Agilefant has aimed to support iterative and incremental development work, help with business-alignment, e.g., through linking daily work items with business level goals, and support managing the developers' efforts as an explicit portfolio.

At the time of writing this, Agilefant implements most of the concepts presented in Figure 4.7, Figure 4.8 and Figure 4.9, as well as much of the related functionality. At the end of 2009, Agilefant was in use in roughly a hundred organizations, with an estimated 500-1000 person global

[45] Since late 2009, three of the companies we were co-operating with started evaluating commercial tools (namely, VersionOne™, Accept™ and Rally™) that to at least some degree possess features for creating and maintaining systems of hierarchical goals (see Figure 4.8). However, all of the above mentioned solutions have been viewed to be relatively complex to use, contain many unnecessary features and be quite expensive. Currently, the author is not, besides Agilefant, aware of open source tools that would contain capabilities for creating and maintaining hierarchical goals.

[46] See http://www.agilefant.org for more information and downloads

[47] See acknowledgements

user base. The majority of the adopting organizations we know of are small (that is, under 50 people).

At the time this summary was written there were – besides publication VI – no published, peer-reviewed results concerning the adoption of Agilefant, or any other tool supporting a hierarchical work item structure for that matter. Thus, further discussion of the possible benefits of and motives for adopting such tool support are out of the scope of this thesis. Instead, the discussion below is limited to describing those features of Agilefant that support the framework described in section 4.5.1. These are:

1) Planning using a system of hierarchical goals/work items (Figure 4.10, ATMAN)
2) Viewing how daily tasks contribute to the high level goals (Figure 4.11, ATMAN)
3) Tracking the progress of a release worked on by more than one team (Figure 4.12 and Figure 4.13)
4) Managing a portfolio of development activities (Figure 4.14 and Figure 4.15)
5) The daily work view to summarize an individual developer's (or a team's) work across several ongoing activities and help select the next task to attend to (Figure 4.16)

The examples below have been taken from real cases of using Agilefant. Most of these (all except Figure 4.15, which comes from the case organization Hector – see Table 5 in section 3.2 on page 60) come from the context of the ATMAN research project at the Helsinki University of Technology (known as Aalto University since the beginning of 2010). ATMAN was a research project of roughly one million euro in total budget and roughly 16 000 man-hours in effort, spanned over three calendar years and employed a total of eight different people (team size peaked at 7 people in Summer 2008) during its course. ATMAN was conducted in collaboration with seven industrial and two research partner organizations. The progress of ATMAN's high level goals (see Figure 4.10 below) was reviewed from two to three times per year by the project's steering group, which decides on the project's continued funding.

In the examples below, ATMAN involves thirteen people and three teams: a team of four researchers, a team of two research assistants responsible for developing Agilefant, and a team of seven students that develop Agilefant under the supervision of the two-person research assistant team. The research team has a diverse 'development portfolio', consisting of writing scholarly publications and theses, industrial collaboration, post-graduate studies, software development, and consulting, with the latter three do not directly relate to the ATMAN project. All of these activities were managed using Agilefant. Coupled with the duration and size of the project, the ATMAN case presents a considerable challenge to deal with using either traditional project management tools like Microsoft Project™ or a simpler, spreadsheet-based backlog management system. Thus, the author considers this example to provide a reasonable proxy for a small software development organization for the purposes of demonstrating how Agilefant supports the framework described in section 4.5.1.

Figure 4.10 Refining a high level goal into smaller work items using the story tree

Figure 4.10 displays the *story tree* of the ATMAN project, representing a view into the product backlog that makes it visible how large, vague work items have been refined into smaller work items[48]. Figure 4.10 lists all the major deliverables planned for ATMAN, for example one doctoral dissertation, 11 international research publications, four master's theses, and so on. The contents of the first listed deliverable –this dissertation – have been expanded to be visible. Comparing with the framework presented in Figure 4.8, the deliverables on the root level of the tree would correspond

[48] Unlike in the more traditional list view into the product backlog, the vertical position of work items in story tree view does not denote relative priority.

to “business goals”. The items one step down from the root level correspond to ‘epic’ and ‘feature’ level goals of Figure 4.8, depending on the size item in question (ATMAN’s iteration length was four weeks). The three work items displayed under “Chapter 4: Results ver 1.0 have been written” correspond to the ‘story’ level goals. The  icon denotes a work item that has been done, while similar ‘S’ and ‘N’ icons denoting that the work items in question have been started, or not started, respectively.

The screenshot has been taken in February 2010 when the first version of the Chapter 4 (*Results*) of this summary was being finalized. The view shows that overall, the Results chapter has been worked on in the December and January sprints, and is planned to be finished during the February sprint – along with several other chapters, as well as delivering a version of the entire thesis document for the supervisor to review. With the exception of the four last ‘feature’ level items displayed, the items that make up the ‘business goal’ of one Ph.D. have been planned to be tackled during the upcoming Feb and March 2010 sprints. If we define ATMAN’s steering group meetings as ‘releases’, this can, according to the definitions provided as an answer to *RQ2* be thought of as ‘release planning’. Because of the large size of the root level items displayed in Figure 4.10, roadmapping them more closely than just denoting that they will be completed at some time later during the course of the ATMAN project would yield little value or meaning.

Figure 4.11 below displays a screenshot of Agilefant’s iteration view to demonstrate how individual tasks contribute to higher level goals.

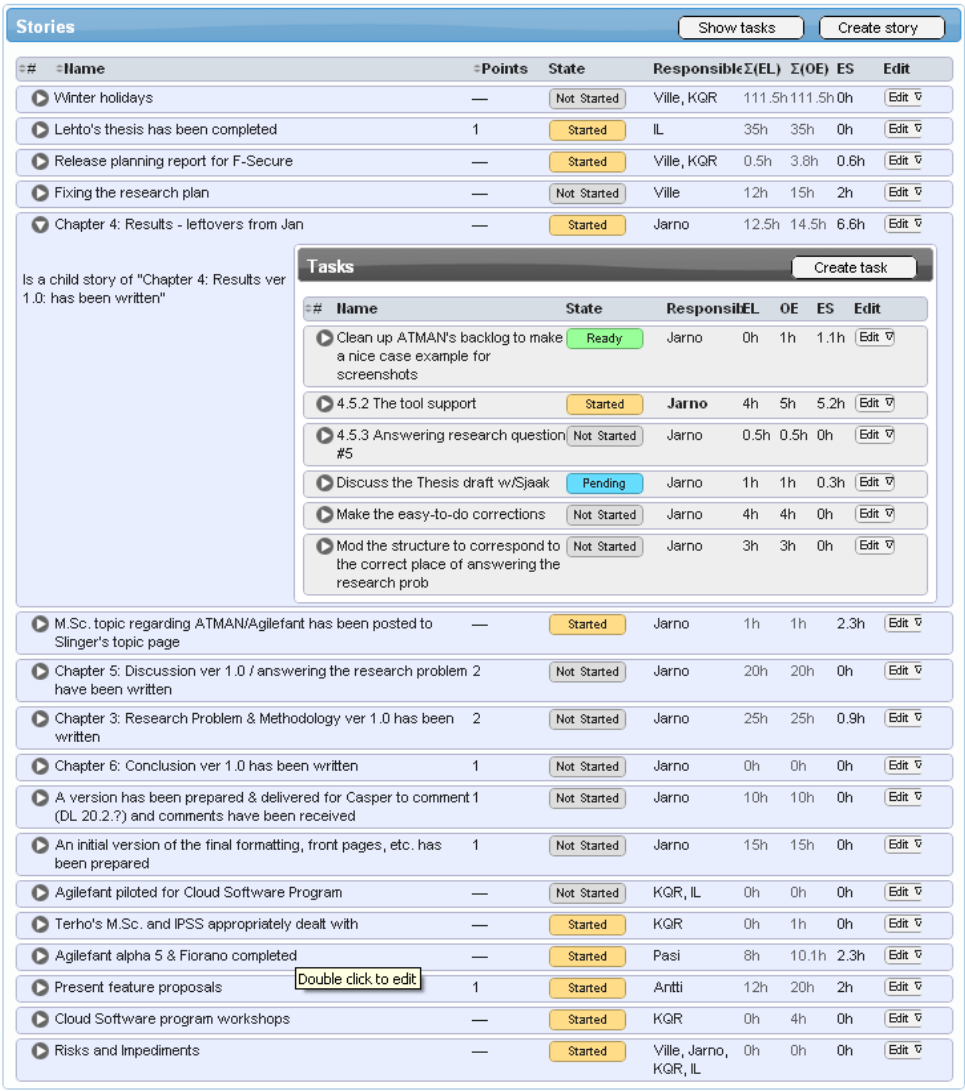


Figure 4.11 Daily tasks' relationship to higher level goals is visible from the iteration view

Figure 4.11 displays the stories ATMAN's team of researchers has planned to get done during the 'Feb 2010' sprint. The tasks for the story regarding the completion of Chapter 4 ("Chapter 4: Results – leftovers from Jan") have been expanded to be visible. Its parent story ("Chapter 4: Results ver 1.0 have been written") is also visible in Figure 4.10. Looking at the tasks, at the time of the screenshot in Figure 4.11 was taken this particular section (4.5.2) was being worked on. Based on the metrics shown the particular task seems to be proving somewhat more laborious than initially estimated (estimated effort left 4h, original estimate 5h, effort spent so far 5.2h).

Figure 4.11 also displays a story related to the development of the next version of Agilefant ("Agilefant alpha 5 and Fiorano completed"). Agilefant's development is managed in a separate backlog, and at the time of the screenshots were taken, it was being concurrently developed by two teams, 'Daytona' and 'Fiorano', with team Daytona also having the product owner responsibilities

of team Fiorano team. Figure 4.12 below shows how Agilefant displays the progress of a release that is being worked on by these two teams.

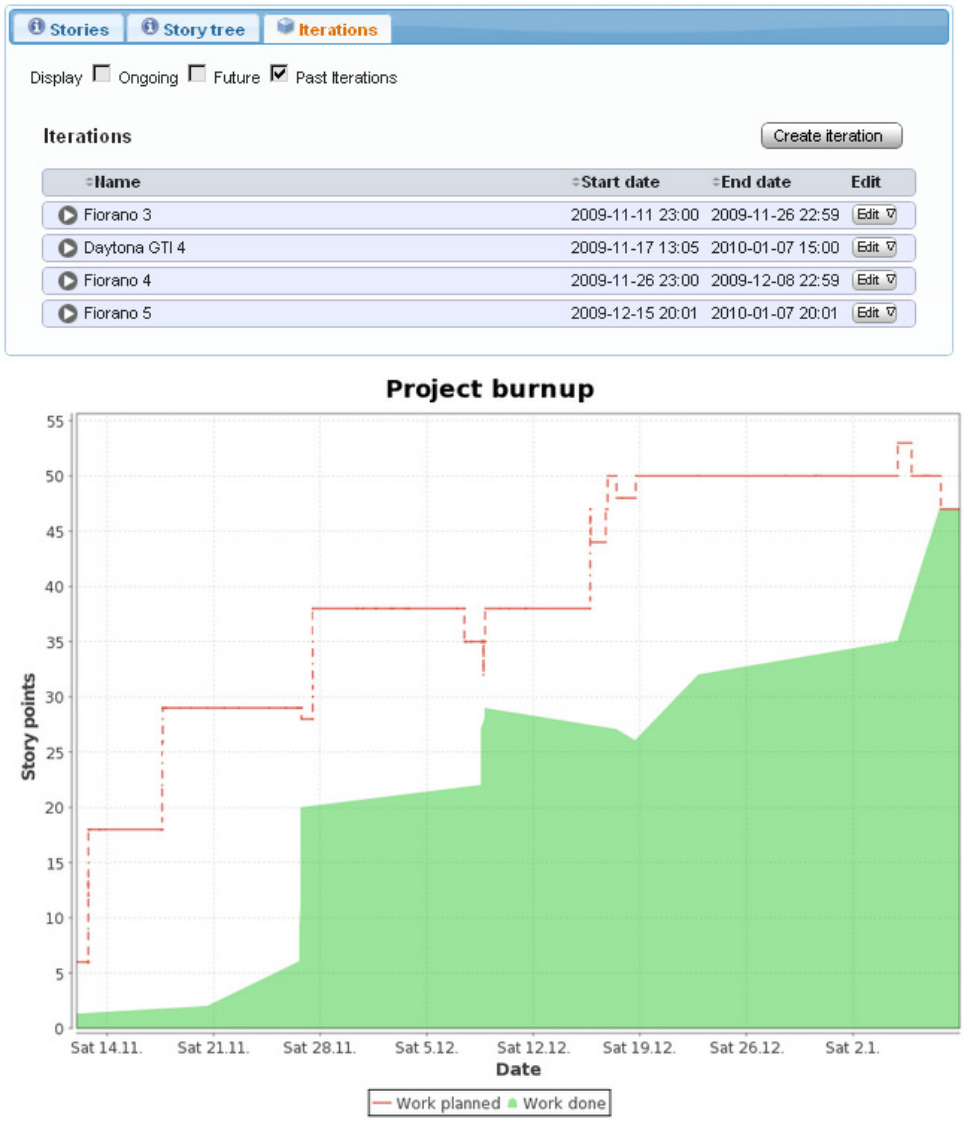


Figure 4.12 The project burn-up and the iterations of a completed release in Agilefant

Figure 4.12 displays the project burn-up Alpha 4 release of Agilefant as well as the past iterations. The four iterations displayed have been named after the two teams working on the release, Fiorano (a seven person student team) and Daytona (a team of two research assistants). Fiorano works in iterations of two weeks, with a total of three iterations planned for the Alpha 4 release. Daytona's "iteration" roughly equals the length of the entire Alpha 4 release. The project burn-up at the bottom of Figure 4.12 displays two graphs denoting the planned scope and progress of the release as the function of time. The red line denotes the scope of the release as the total sum of story points from those work items that are planned to be completed in the release (that is, those work items that

reside in the product backlog in the section corresponding to the Alpha 4 release or in any of its iterations). Thus, increases in the red line mean a scope increase of the project. Due to the progressive refinement of requirements, such 'increases' are a natural consequence of splitting large, vague user stories with possibly no estimates to begin with, into smaller items that do have estimates. The green area denotes the story point sum from those work items that both are planned to be completed in the release and have been marked as *Done*. As the work items planned for the release get done (the height of the green area goes up) or are moved out of the scope of the release (for example, it is realized that a work item cannot be completed in time or it simply is no longer needed; in these cases the red line goes down), the two graphs get closer and, in the case of a successful scoping of a release, eventually meet. Note that the height of the green area may also go down (as seen above) as something that was thought to be 'Done' actually is found to be otherwise.

Figure 4.13 below shows the story tree of the Alpha 4 release of Agilefant 2.0[49], displaying all the work items that at the end of the release were planned to be included.

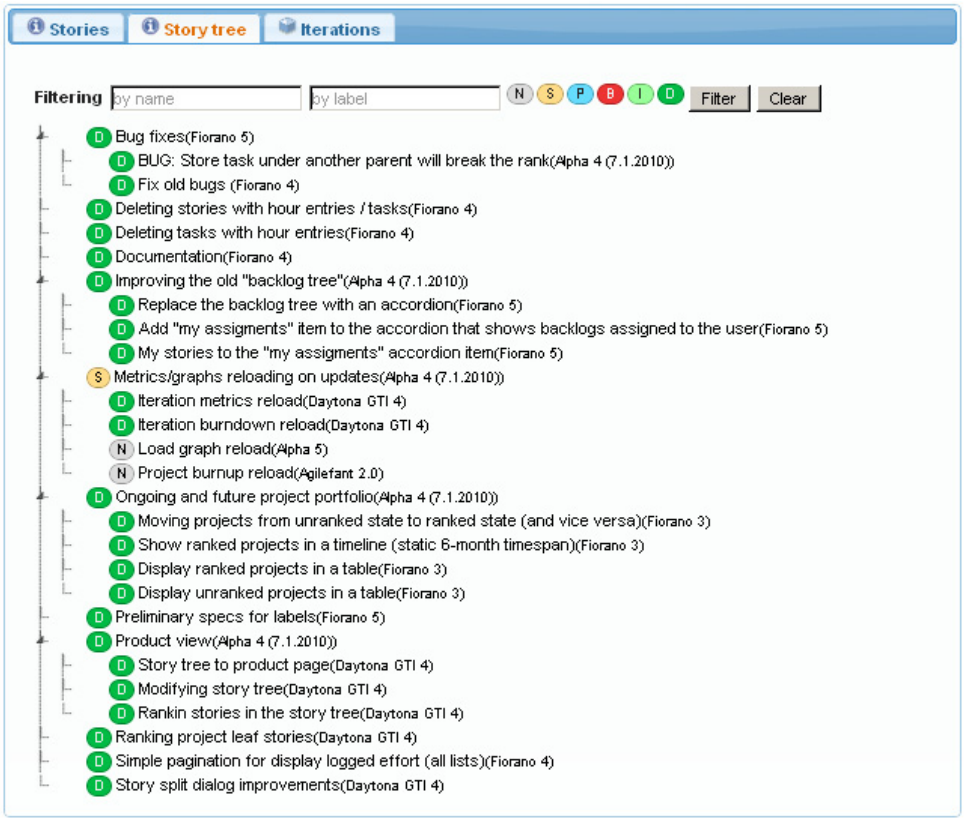


Figure 4.13 The story tree for Agilefant's Alpha 4 release

Two of the planned work items shown in Figure 4.13 could not be completed in time. One of these ("Load graph reload", marked as [N]ot started) was moved to be done in the Alpha 5 release, and the other ("Project burnup reload") some time later but still in time for Agilefant 2.0. The story tree also displays in which iteration the story got done, as well as due to naming the iterations according

[49] Agilefant 2.0 has been modelled as a 'Product', while the different Alphas are 'Projects'.

to the teams in question, which team was responsible for the story.

Figure 4.14 below shows Agilefant's view for managing the portfolio on current and immediately upcoming activities that require attention from the development. This view displays the general status of the ongoing activities (denoted as a traffic light set by the product owner of the activity in question). Ongoing and immediately upcoming activities can be prioritised (rank-order) against each other, and resources can be allocated ('Assignees'). The portfolio view has been designed to support the work of the portfolio council and the traffic control squad (see Figure 4.9 and section 4.5.1 on pages 91-93).

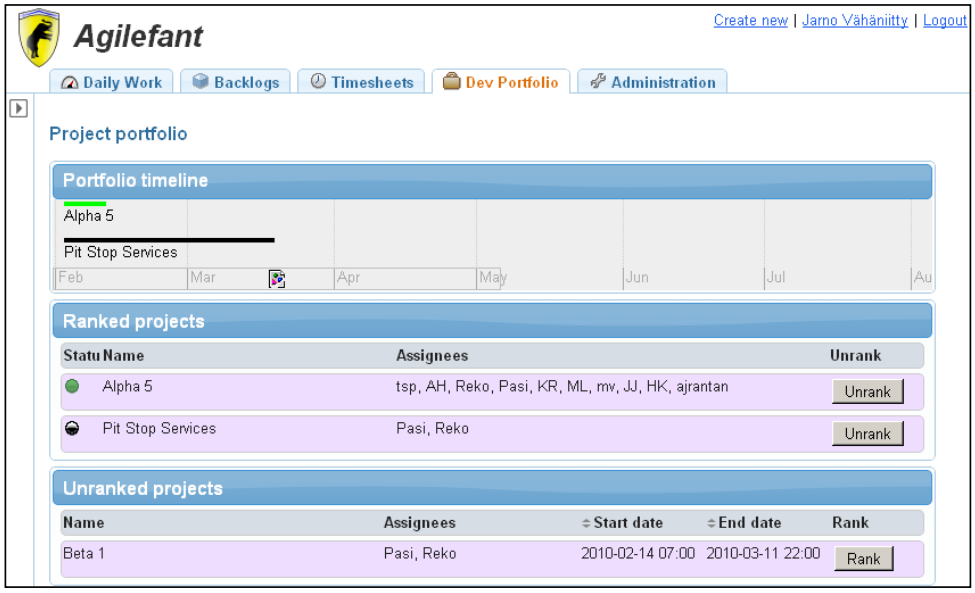


Figure 4.14 The development portfolio view in Agilefant (referred to as the 'project portfolio')

The simple case shown in Figure 4.14 is the portfolio of activities that the Daytona team (the two research assistants responsible for developing Agilefant) has to attend to. It displays the ongoing Alpha 5 release, the Beta 1 release that is planned to follow it, as well as a longer activity called "Pit Stop Services", which refers to support requests by organizations that are currently using Agilefant. Alpha 5 has been ranked as more important than Pit Stop Services. While the set priority is not absolute in the sense that all of the work in Alpha 5 would be more important than any of the support requests that might come up, explicitly agreeing on the activities' priority helps in making trade-offs when necessary. Alpha 5's green colour denotes that the release is, as judged by the product owner, proceeding as expected with no scope-outs necessary. The black colour of Pit Stop Services indicates that the activity is not managed using Agilefant on a detailed level, and thus, its status cannot be deducted from the system. This way, even though not all of the activities that require attention from development must or even should be managed using detailed stories and tasks, all of them (as well as the people assigned to them) can still be represented in Agilefant. Also, should team Daytona need to track how they are spending their time, they can log spent effort directly to the Pit Stop Services activity itself.

While Figure 4.14 illustrates the basics of managing a development portfolio using Agilefant, the development portfolio in most small software development organizations is based on publications IV and V (as well as existing literature – see sections 2.3 and 2.4) likely to be much more complex.

As an example of this, Figure 4.15 below displays a snapshot of the development portfolio of Hector (one of the case organizations studied for publications IV and V) from 2010. The list of 'Unranked projects' continues beyond the bottom edge of the picture, with the total number of activities in both lists some 40 activities. The total number of employees in the organization is 30. The real name of the organization (as well as the names of its clients) have been blanked out from Figure 4.15) for confidentiality reasons.

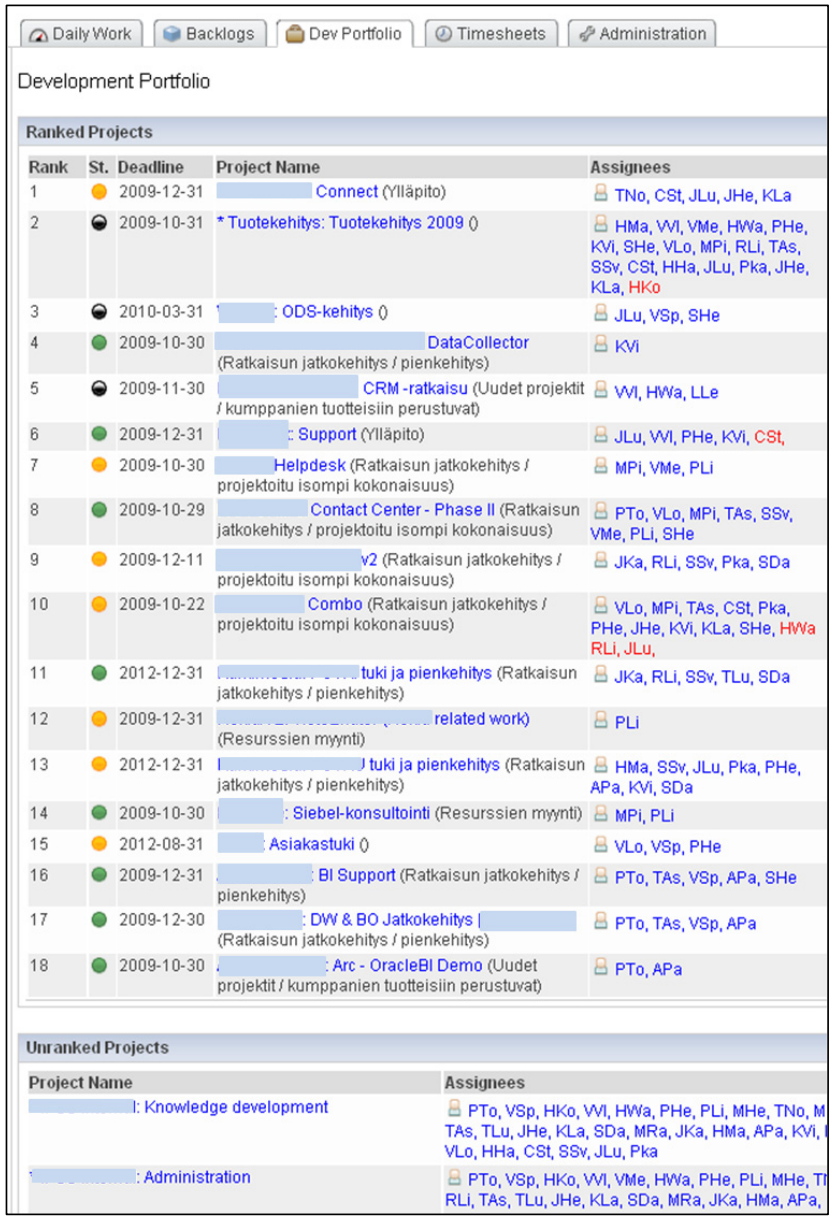


Figure 4.15 Ranked projects in the development portfolio of case Hector

As the final example of how Agilefant supports linking agile software development with product

and portfolio management, Figure 4.16 displays a screenshot from Agilefant's *Daily work* view.

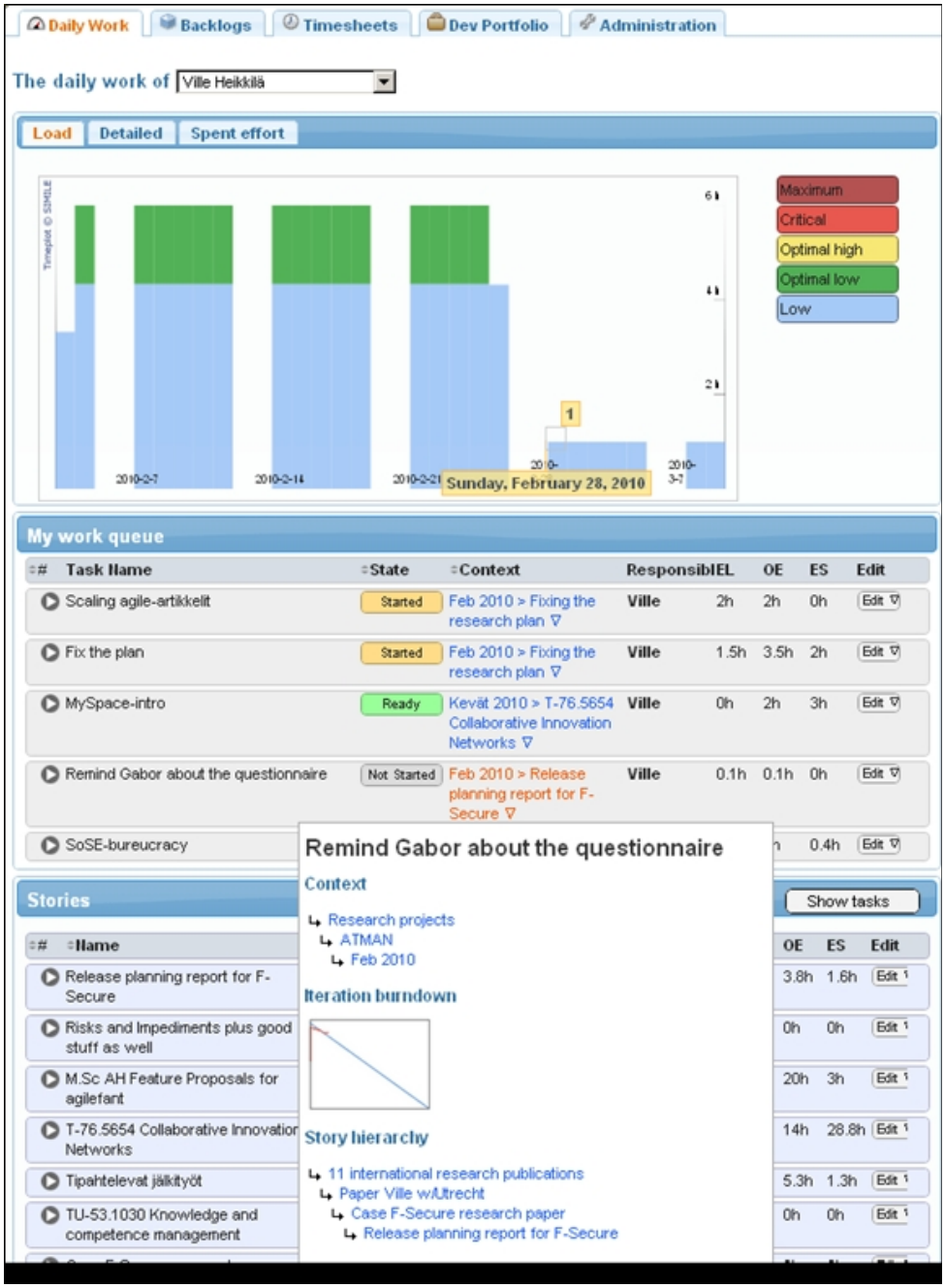


Figure 4.16 The Daily work view summarizes and helps organize the duties of an individual across his activities

The daily work view (Figure 4.16) is designed to help individuals deal with the "bottom level of portfolio decision-making" depicted in Figure 4.9 in section 4.5.1 (p. 93). It summarizes the duties

of a user from all of the activities he is currently involved in, gives a rough idea of his current and upcoming workload, and provides a 'work queue' for selecting and organizing the task(s) to be attended to next in the bustle of multiple roles and responsibilities often present in small organizations developing software (see publications IV and V). These are below explained in more detail.

The *Load* section at the top of the page calculates the total amount of effort required to complete the tasks the person is either directly (via having assigned as one of the people responsible for a given task) or indirectly (via being assigned to a project or an iteration containing tasks not assigned to anyone) responsible for, and compares the total against set thresholds. The middle section of Figure 4.16 displays the person's *Work queue*, which is a personal queue for choosing and ordering those tasks that currently seem the most relevant to attend to from the perspective of the individual. A task's priority in the work queue is irrespective of its "real priority", which can in principle be deducted from the activity's and the stories' relative priorities. We propose that there is always room for intuition in picking the "next most relevant task"; depending on e.g. the mood of the individual, or then simply the time available before e.g. the lunch break, a meeting, or heading home. The work queue may also help various stakeholders (for example, the business owner or the member of another team dependent on this team's progress) to understand the progress (or non-progress) of the activities they are interested without disturbing the responsible person.

The lower section of the daily work view lists all of the not-done stories currently relevant to the person, either via being directly responsible for the entire story, or some of its tasks only. The context (i.e. the iteration and the activity it belongs to) of each task are directly visible from both of the work queue and the story list, and a pop-up can be opened to display the related story hierarchy as well as the progress of the iteration in question. For example, the pop-up in Figure 4.16 displays how the very small task "Remind Gabor about the questionnaire" contributes both to the publication and the international research co-operation goals set for the ATMAN project.

Note, that in the hypothetical case of by-the-book-Scrum, the daily work view would be of little value. For an individual developer, there would be only a single backlog – that of the currently ongoing iteration – to pick tasks from. Based on the publications included in this dissertation as well as the examined related work, however, it would seem that vast majority of organizations actually are – as well as will in the near future be – in the process of adopting agile software development for only some of their activities, rather than practicing "pure Agile" for everything. Also, it seems there are nearly always multiple ongoing activities that require the development resources attention. Thus, it is plausible that the Daily work view would be relevant in practice.

4.5.3 Limitations

The most important identified limitations of the research methods used for answering RQ5 are highlighted in Figure 4.17 and discussed below.

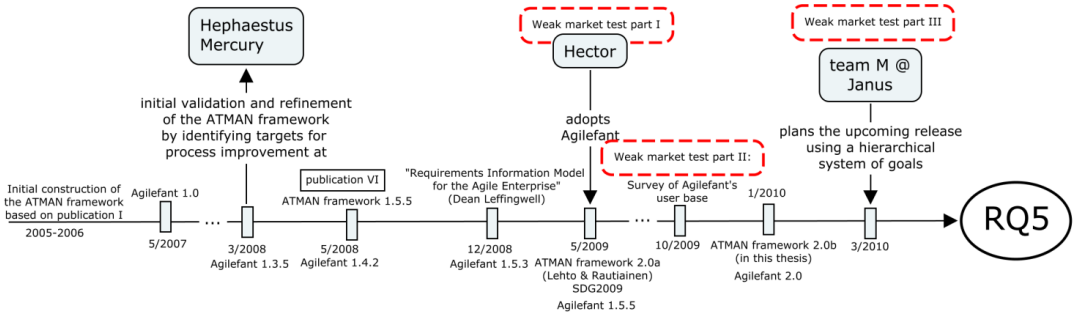


Figure 4.17 The most important identified limitations of the methods used for answering RQ5

First, we discuss the *market testing* of the constructed framework. A construction is considered to have passed a *weak market test* if at least one manager responsible for his or her financial result has been willing to use the solution (Lukka & Kasanen 1995). Thus, if an organization adopts a support tool that complies with the framework, for example, Agilefant and utilizes its capabilities for managing the development portfolio and linking long-term product plans with the developers' daily task, the framework constructed to answer RQ5 could be considered to have passed the weak market test. In contrast, passing a *strong market test* (Kekäle 2001) would require having evidence of economically measurable improvements due to the adoption of a support tool that complies with the framework in one or more of the adopter organizations. This dissertation makes no claims towards whether the construction passes the strong market test, but in the following we discuss the evidence in support of passing the weak market test. The evidence for passing the weak market test is available from several sources: the research team the author himself belongs to, case organizations Hector and Janus at which Agilefant was adopted during the Spring of 2009, a survey of Agilefant's users in 10-11/2009, independent reviews and assessments of tools for backlog management, and the adoption rates of other existing support tools that comply with our framework.

The team of researchers that the author himself is a part of has since the Fall of 2007 used Agilefant to manage their own research activities. Although we have established and maintain the business-development linkages as described in framework, due to researcher bias this line of evidence does cannot be considered very strong in terms of evaluating whether the framework passes the weak market test.

At Hector, all of the work performed by the employees has from 5/2009 to present (4/2011) at least somehow been accounted for in Agilefant. Agilefant allows, but does not impose on applying the frameworks described in section 4.5.1. For example, Agilefant allows for activities having different degrees of detail in terms of e.g. having distinct stories and tasks, and logging spent effort on the level of individual tasks. On the other hand, if detailed story or task breakdowns – or even distinct iterations – are not needed, using them is not necessary. As another example, if spent effort is to be recorded, it can be logged directly to the activity itself instead of otherwise unnecessary creation of stories and tasks for storing the spent effort information. While these features are crucial for making it possible to use Agilefant to manage the kinds of "project portfolios" that according to publications IV and V may be typical for small software organizations, the downside is that having an organization adopt Agilefant does not guarantee that the weak market test would be passed. To the knowledge of the author, none of the ongoing activities at Hector have to date (4/2011) fully used Agilefant's capabilities for linking development tasks to long-term product plans. However,

they are using Agilefant to manage their development portfolio (Norja 2010)[50].

Another line of evidence was provided by the first annual Agilefant user survey conducted in 10-11/2009. In total, 133 invitations to participate in the survey were sent by email. We received 35 answers. According to the responses, versions of Agilefant based on the framework described in publication VI or later have been adopted by at least some 30 teams in some 20 organizations worldwide. If we take the assumption that the numbers of teams and organizations of Agilefant adopters yielded by the survey in Finland are "exact", we can, based on the number of visits to Agilefant.org from Finland vs. the rest of the world extrapolate a rough number of the total amount Agilefant adopters worldwide: 500+ users in 100 teams and 50 organizations. However, as we do not know how the organizations are using Agilefant, this line of evidence, does not help in evaluating whether the framework pass the weak market test either.

As another possible line of evidence, a version of Agilefant based on the framework described in publication VI received in 9/2009 a favorable review in an assessment of the "most compelling open source options for agile project management" by an independent consultant[51]. Of particular interest in this context is the notion presented in the review that of the assessed solutions, Agilefant was the one best suited for large projects and organizations. This is most likely due to the fact that the other solutions do not account for the multi-project context most real organizations face.

As the fourth line of evidence, the author interviewed in March 2010 the product owner of a relatively independent ten-person team at the case company Janus regarding whether he was applying Agilefant's capabilities for creating and maintaining hierarchical work breakdowns in backlog management. In the first part of the interview, the product owner had, independently of the author's influence created some hierarchical structure in planning the contents of an upcoming major release. At the end of this first interview, the author asked whether the product owner would see value and be interested in structuring the entire product backlog for his product, as well as the upcoming release into a hierarchical system of goals as prescribed by the framework. For the second part of the interview, conducted two days later, the product owner had spent a total of "less than two hours" for organizing the product backlog, including the contents of upcoming the release into a hierarchical system of goals. He perceived that the resulting structure allows for easier communication of the release contents as well monitoring its progress once the project is underway. The product owner also agreed to hold short weekly internet telephone meetings where he will show and discuss the status of the release project to the author. This suggests that the framework does pass the weak market test. However, the evidence backing this claim is not published or otherwise available to public, it is questionable whether this can be used as evidence whether the construction passes the weak market test or not.

As the final line of evidence, a tool called VersionOne™ seems, after spreadsheets (see discussion on spreadsheets versus dedicated tools for managing agile software development in section 2.2.13) to be currently at least among the most adopted support tools managing agile software development. VersionOne™ seems, at least on the surface, to be very much in compliance with our framework. While the "State of Agile 2010" survey[52] that places VersionOne™ as the most adopted agile development tool is conducted by the tool vendor itself, making its validity subject to question, other sources that to our knowledge are independent also mention VersionOne™ as at least one of the more popular tools on the market currently (Schiel 2009, Goth 2009).

[50] Norja's master's thesis (2010) is available for download at least at http://tinyurl.com/mscnorja.

[51] http://olex.openlogic.com/wazi/2009/comparing-open-source-agile-project-management-tools/

[52] http://www.versionone.com/state_of_agile_development_survey/10/default.asp

Taken together, the above lines of evidence would seem to support the notion that the framework does pass at least the weak market test.

4.5.4 Answering research question #5

Having described an updated version of the framework for agile software development with product and portfolio management and the current state of the tool support initially presented in publication VI, RQ5 (*How can long-term product and business planning be linked to daily software development tasks in agile software development? What role does portfolio management play here?*) is answered as follows:

The high-level goals (e.g. vision, business goals and epics) are, via the processes of roadmapping, release planning refined into short-term objectives (e.g. features and stories), which in turn get expressed as actionable tasks via iteration planning. The trace of how the "smaller" work items have been refined from the "larger" work items is kept, resulting in a hierarchical system of goals. This resulting system can then be explored bottom-up or top-down as needed. If a high-level goal changes or is dropped, the entire tree should be examined and altered and/or pruned as necessary. Also, the process of refining large work items or goals into smaller ones may, especially as development proceeds, yield new information that results in modifications to the original high-level work item or goal.

Portfolio management becomes crucial when there is more than one initiative that requires attention from the same resource pool. However, there is not a singular process of "portfolio management", but multiple levels of portfolio decision-making: at the highest level, portfolio decision-making is concerned with deciding on the set of products and services offered and developed by the organization, as well as deciding on the relative spending across the set of business areas. This is commonly referred to as *product portfolio management*. In contrast, d*evelopment portfolio management* deals with tactical prioritization and resource allocation as well as resolving resource conflicts and emergencies across the set of possible activities that in principle compete for the same pool of resources. And, in *daily work*, the development people choose which task(s) and from which activities next get attended to.

If the activities are planned employing a hierarchical system of goals – whether they follow an agile life cycle or not – business priorities in an organization become more transparent. Thus, decisions can be taken more and more on the level where the actual work is being done, and fewer escalations are needed. It is plausible that this would result in improving both the speed of decision-making as well as the quality of the decisions themselves.

Based on our review of the related work, there currently does not seem to be a consensus on whether creating and maintaining hierarchical work item structures is necessary or even beneficial (see sections 2.2.10 and 2.2.11). What, however does seem obvious is that maintaining multiple systems of hierarchical goals (that is, there would be one for each activity – and even in small software organizations there are more ongoing activities than people) by hand is arduous, if not impossible. Thus, we propose the possibility that even in small software organizations, proper tool support may be crucial in linking agile software development with product and portfolio management. In line with this proposal, at least some of the commercial tools on the market for agile software development work item management do seem to support the creation hierarchical work item structures.

5 Conclusion

This chapter concludes the summary part of this dissertation. First, the limitations that apply to this study are summarized (section 5.1), and the research problem is answered in the light of the most important identified limitations (section 5.2). The theoretical contribution of this dissertation is compared with related work (section 5.3) and summarized (section 5.4). Then, the practical implications of the results are presented (section 5.5). The chapter finishes with providing possible directions for future research (section 5.6).

5.1 Summary of the limitations

Summarizing from the identified limitations concerning the answers to the individual research questions (see sections 4.1.3, 4.2.3, 4.3.3, 4.4.3 and 4.5.3 for more details), two are salient from the perspective of answering the research problem.

The first key limitation lies in the fact that we do not have evidence of how representative the studied software organizations are of the population of small software development organizations. The needs for more explicit long-term product and business planning as well as portfolio management seem to be present at least in companies that a) are growing and/or seek growth and b) strive to adopt agile software development for at least some of their activities. However, even limiting the answer to the research problem to apply for only such companies may not be sufficient. This is because all of the studied organizations have, at one point or another sought the advice of the author and his colleagues with respect to the subject matter of this study and have been willing to invest their time as well as funds in participating our research projects. From the perspective of adopting and applying the suggestions, models and frameworks created during this study, such organizations possess may possess characteristics that set them apart from the rest of the intended population. It could even be argued that it is because of these characteristics, rather than because of the possible value in the presented approaches, models and frameworks themselves, that the organizations perceive the collaboration and its results as valuable. While the organizations have made progress during the collaboration, it may well be that other suggestions, approaches or frameworks could have yielded the desired (or even better) improvements. Except for comparing our results with the advice and frameworks presented in current literature discussing product and portfolio management in conjunction with agile software development as we have done throughout chapter 4, we have little to offer in terms of how this could be evaluated.

Indeed, the collaborative setting in which the vast majority of the research for this dissertation was conducted is the source of the second key limitation. Both the researchers and the involved company personnel have vested interest in producing good results in the form of new approaches, frameworks as well as in applying them in practice. Because this ‘bias’ is inherent to the action research approach, there is little possibility to completely avoid it, and it can also be considered a pre-requisite for this kind of research to be conducted in the first place. Still, it must be noted that aside from the description of the research methods and limitations presented in this summary, we on the whole have little knowledge of, for example, how lasting the changes resulting from the collaboration were.

Thus, we summarize the key limitations of this study as follows:

> *For small organizations developing software and intent on*
>
> *1) growing,*
> *2) applying agile software development methods in at least some, but not necessarily all of their activities, and*
> *3) being open for process improvement collaboration with outside experts,*
>
> *the approaches and frameworks presented in this dissertation seem to – at least temporarily – alleviate the problems the organizations' personnel perceive as being related to a disconnect between business and development decision-making.*

5.2 Answering the research problem

Before answering the research problem, we below summarize the answers to the research questions.

RQ1 (*What are the key processes and decision areas that connect Business with Iterative and incremental software development?*) is answered as follows:

> The key decision areas that connect top management and iterative and incremental software development are *portfolio management, organisation*, *development model*, *product management* and *quality strategy*.
>
> The key processes that should connect business and development decision-making are *development portfolio management* (part of the portfolio management decision area), *roadmapping* and *release planning* (parts of the product management decision area).

RQ2 (*What does roadmapping mean in the context of agile software development and how can a roadmap be visualized?*) is answered as follows:

> In the context of agile software development, *roadmapping* refers to grooming the product backlog so that the high level goals for the foreseeable future are visible and reflect the current understanding of the relevant stakeholders. In contrast, *release planning* refers to the planning and continuous refining of the contents of the immediately upcoming release.
>
> In this context, the roadmap itself should be considered as a *view* into the product backlog that depicts how a particular solution (or line of solutions) is currently planned to evolve in the foreseeable future. It shows the release dates, the planned features and their estimated sizes, services that demand the developers' attention, and the planned resource usage.
>
> Ideally, it is possible to generate or otherwise a visual roadmap directly from the product backlog itself because roadmaps as separate documents tend to quickly get out-of-date. An example notation for visualizing roadmaps is depicted in Figure 4.3 on page 84.

RQ3 (*Do small software organizations suffer from the lack of explicit portfolio management?*) is answered as follows:

> Small software organizations seem to suffer from the lack of explicit portfolio management at least when the following conditions hold:
>
> 1) The organization is leveraging customer-specific projects for product development,

2) The development personnel possess multiple roles and responsibilities and are concurrently performing many different types of activities
3) The organization has recently grown, or the management intends to grow it in the near future

RQ4 (*How can portfolio management be set up and conducted in small software organizations striving for agile software development?*) is answered as follows:

The key steps in setting up and performing development portfolio management in small software organizations are:

1) Appointing a group of people to be responsible for portfolio decision-making
2) Building a publicly visible list of all ongoing activities that require time from development, including the information on who are assigned to which activity
3) Synchronizing the portfolio
4) Meeting regularly at portfolio synch-points to keep the list of ongoing activities up-to-date, perform short-term prioritization and setting the default resource allocation until the next meeting
5) Agreeing on how decisions affecting more than one ongoing activity are to be made in 'emergency' type situations
6) Identifying the different types of development activities, setting target spending levels that reflect the organization's strategy and tracking the spending, either according to actual or planned resource consumption
7) Setting a limit to the number of concurrent activities a team and/or person can be involved in
8) Tracking the development teams' / developers' workload
9) Ensuring that incentives do not steer people towards local optimization

RQ5 (*How can long-term product and business planning be linked to daily software development tasks in agile software development? What role does portfolio management play here?*) is answered as follows:

Roadmapping, release planning and iteration planning refine high-level goals (business goals and epics) are refined into short-term objectives (features and stories), which in turn get expressed as tasks. The trace of how the "smaller" work items have been refined from the "larger" work items is kept, resulting in a hierarchical system of goals per ongoing development activity.

Portfolio management becomes crucial when more than a single initiative requires attention from the same resource pool. Instead of a singular process of "portfolio management", there are multiple levels of portfolio decision-making: *product portfolio management* is concerned with deciding on the set of products and services offered and developed by the organization, as well as on the relative spending across the set of business areas. *Development portfolio management* deals with tactical resource allocation, prioritization and conflict management across the set of possible activities that compete for the same pool of resources. And, in *daily work*, the development people choose which task(s) from which ongoing activities get next attended to.

We propose that employing a hierarchical system of goals makes business priorities more transparent across the entire organization. As a result, decisions can be taken nearer to the level where the actual work is being done, which improves both the quality and the speed of decision-making. Proper tool support may be important for linking agile software development with product and portfolio management even in small organizations, because

maintaining multiple systems of hierarchical goals by hand is arduous.

The answers to the research questions are also summarized in Table 8 below.

Table 8 Summary of the answers to the research questions

Research question	**Answer**
RQ1: What are the key processes and decision areas that connect Business with Iterative and incremental software development?	The key decision areas are *portfolio management, organisation*, *development model*, *product management* and *quality strategy*. The key processes are *development portfolio management, roadmapping* and *release planning*.
RQ2: What does roadmapping mean in the context of agile software development and how can a roadmap be visualized?	*Roadmapping* means grooming the product backlog so that the high level goals for the foreseeable future reflect the current understanding of the relevant stakeholders. In contrast, *release planning* refers to the planning and refining of the contents of the immediately upcoming release. The *roadmap* is a view into the product backlog that depicts how a (line of) solution(s) is planned to evolve in the foreseeable future.
RQ3: Do small software organizations suffer from the lack of explicit portfolio management?	Yes, at least when customer-specific projects are leveraged for product development, the personnel possess multiple roles and responsibilities and are concurrently performing many different types of activities, and organization growth is imminent (or recent).
RQ4: How can portfolio management be set up and conducted in small software organizations striving for agile software development?	The key steps include (e.g.) building and maintaining a public, prioritized list of all ongoing activities, ensuring that incentive systems do not steer towards local optimization, and appointing a group of people to meet regularly to decide on priorities and resourcing.
RQ5: How can long-term product and business planning be linked to daily software development tasks in agile software development? What role does portfolio management play here?	The trace from iteration level work items and high-level product and business goals must be retained. The resulting hierarchical systems of goals make the business priorities more transparent, which speeds decision-making. However, achieving it may require tool support. There are multiple levels of portfolio decision-making moderating the flow of strategy to action. *Product portfolio management* is concerned with deciding on the set of products and services offered and developed. *Development portfolio management* deals with short-term resource allocation, project prioritization and conflict management. When developers are involved in more than one concurrent activity their *daily work* involves choosing from a portfolio of tasks the next one to work on.

Summarizing from the answers to the research questions, the research problem (*How can product and portfolio management be linked with agile software development?*) is answered as follows:

> Linking product management with agile software development requires changes in *roadmapping* and *release planning*. Rather than a separate artifact, the roadmap should be considered as *a view into the product backlog* that on a high level depicts the plan for the foreseeable future. Roadmapping then means simply the continuous grooming of the product backlog to reflect the current understanding of the relevant stakeholders. In contrast, release planning refers to the continuous planning and refining of the contents of the immediately upcoming release.
>
> Rather than being a single high-level process, portfolio management in the context of agile software development requires portfolio decision-making on multiple levels. *Product portfolio management* is concerned with deciding on the set of products and services offered and developed. *Development portfolio management* deals with tactical resource allocation, prioritization and conflict management across the ongoing activities. And, when developers are involved in more than one concurrent activity, their *daily work* involves choosing choosing the most important task to attend to from possibly multiple different backlogs. Effective portfolio decision-making on the various levels requires that business priorities are transparent. To achieve this, the trace of how high-level goals (business goals and epics) have been split into short-term objectives (features and stories) and on to actionable tasks should be retained. This results in a hierarchical system of goals per activity, the maintaining of which may require tool support.
>
> Explicit portfolio management seems relevant for small software organizations at least when customer-specific projects are leveraged for product development, the personnel possess multiple roles and responsibilities and are concurrently performing many different types of activities, and organization growth is imminent (or recent). The key actions in setting up and conducting portfolio management include building and maintaining a public, prioritized list of all ongoing activities that require the developers' attention, ensuring that incentive systems do not steer towards local optimization, and appointing a group of people to meet regularly to decide on priorities and resourcing. Furthermore, explicit portfolio management may be crucial even for small software organizations.

Taking into account the identified limitations of this study (see section 5.1 on page 109), the above answer is valid at least for those small organizations that are intent on growing (or have recently grown), applying agile software development methods in at least some, but not necessarily all of their activities, and being open for process improvement collaboration with outside experts. While we make no claims towards this, the above suggestions may, based on a systematic review of related work, apply to other software organizations intent on lean/agile transformation as well.

5.3 Comparison with related work

Both publications I and II as well as the reviewed literature (see sections 2.2.1-2.2.4) identify *roadmapping*, *release planning* and what in this dissertation is referred to as *development portfolio management* as the key processes for linking product and portfolio management with software development. However, based on a review of the literature, the nature of these processes in the context of agile software development is not well understood. Practitioners also seem to view this area as one of the most challenging in managing software development (see section 2.2.5).

Drawing from publication III and the reviewed related work (see section 2.2.12), we provided an example of roadmapping from the small software organization context as well as formulated "agile compatible" definitions for 'roadmapping' and 'release planning' as an answer to RQ2. Judging

from both the reviewed literature (see sections 2.2.3-2.2.5) as well as publication III, preparing and updating roadmaps as separate, detailed documents tends to lead into them not being updated. Thus, unless the roadmap can be automatically generated from the product backlog itself, the value of our roadmap visualization from a practitioner perspective lies in providing a visual checklist of what should be considered as part of long term planning.

Existing literature does not discuss whether small software organizations should or should not perform explicit portfolio management. However, based on publications IV and V, explicit portfolio management indeed seems to be beneficial at least when the organization is growing (or intends to grow) and customer-specific projects are being leveraged for product development. While our study does not claim that our case organizations are representative of the larger population of small software organizations, a review of the existing literature (see section 2.4) would suggest that the factors we in publication IV identified as increasing the need for explicit portfolio management may actually be quite common in those small organizations that have enjoyed recent success.

Publications IV, V and VI as well as what little literature has in the recent years addressed portfolio management in the context of agile software development (see section) advocate the iteration-wise synchronization of concurrent activities that require attention from the development. An organization-wide synchronized iteration cadence in turn corresponds to the notion of *portfolio reviews* as the preferred approach to implementing portfolio management in a turbulent environment (see section 2.3.3). In contrast, it seems questionable whether the probably more common *gates dominate* approach to portfolio management can meet real success in the context of agile software development (see section 2.3.4).

The discovered experience reports suggest that long-term planning remains vital even when the software development processes are agile (see section 2.2.4-2.2.5). Our results suggest that strategic ambitions, possible long-term plans and their business implications may not even be visible even in small software organizations if attention is not explicitly devoted to their communication even. In publication VI we advocate using hierarchical work item structures (see sections 2.2.10-2.2.13) to link long-term product and business goals with daily development tasks[53], and introduce Agilefant as a prototype tool support for creating and maintaining such structures. In line with our results, more and more authors are in recent practitioner books and published experience reports starting to address the notion of hierarchical work item structures in the context of agile software development, though the topic remains controversial (see sections 2.2.9-2.2.11).

At the time this summary was written, we were – besides publication VI – aware of other published, peer-reviewed results concerning support tools for managing hierarchical work item structures. It is plausible that maintaining such structures by hand is arduous at best, if not impossible. Indeed, some research articles and experience reports cite lacking work item tool support (spreadsheets) as an obstacle for linking agile software development with business decision-making (see section 2.2.13).

5.4 Theoretical contribution

For a company in the software business striving for lean/agile software development, it seems essential to understand how to link the development process with product and portfolio

[53] The most notable influences since publication VI to the framework as presented in section 4.5.1 are the work of Lehto and Rautiainen (Lehto & Rautiainen 2009, Lehto 2010) and Leffingwell (2011). Lehto further developed the framework from publication VI to more explicitly describe a hierarchy of work items. Leffingwell coined *Epics*, *Features* and *Stories* to denote work items on three abstraction levels in the context of agile software development.

management decision-making. Current literature has mostly discussed what should be done at the "floor level" or in an individual project, leaving the link to business and product management largely unaddressed. This can be especially problematic for small, growing software organizations companies who, in order to remain operationally effective, need to maintain the big picture of the ongoing work of the development staff and align this with the long-term plans of the enterprise.

To help with the above mentioned concern, this dissertation summarizes existing as well as presents new understanding for linking business decision-making with modern development methodologies such as Scrum in the face of the practical realities that apply to at least many small software organizations. We present how the key processes linking business and development decision-making – portfolio management, roadmapping and release planning – should be understood in the context of organizations striving for agile software development in at least some of their activities. We also present an example of how a product roadmap can be visualized, explain that explicit portfolio management is (at least when certain conditions hold) crucial for small organizations as well, and provide guidelines for practicing it. As proof-of-concept, we introduced Agilefant as tool support for creating and managing a portfolio of activities of which some (but not necessarily all) are planned and managed using hierarchical work item/goal structures. Such structures provide transparency to business priorities while still making just-in-time elaboration required by agile methods possible.

This study does not offer evidence on how the results apply beyond the context of small organizations that are intent on growing, applying agile software development methods and being open for process improvement collaboration with outside experts. However, reflecting the results of this dissertation in the light of the recent literature on agile software development, it is plausible at least some of the results could be applicable in larger organizations as well.

5.5 Practical implications

For those who identify their organization and situation as reasonably similar to the case organizations of the included publications, there are a number of practical implications. These are briefly listed in sections 5.5.1-5.5.5 below according to the order of the respective research questions, along with pointers for more detail that can be found in this summary or the included publications.

Many of the lessons learned in this study have been incorporated into Agilefant (www.agilefant.org; see also section 4.5.2 on pages 95-105). It is our sincere hope that as Agilefant's adopter base grows, this dissertation' contributions will more and more be taken up in practice.

5.5.1 Linking business and development

RQ1: What are the key processes and decision areas that connect Business with Iterative and incremental software development?

- The most important processes that should connect Business and Development are development portfolio management, roadmapping and release planning (see publications I and II)
- In addition to the above processes, business priorities and the constraints set by the development should be reflected in the organization structure, the development life-cycle models adopted for different types of activities as well as the quality goals (see publications I and II)

5.5.2 Long-term planning and agile software development

RQ2: What does roadmapping mean in the context of agile software development and how can a roadmap be visualized?

- Roadmapping in agile software development should be understood as grooming the product backlog so that the high level goals for the foreseeable future are visible and reflect the current understanding of the relevant stakeholders; release planning refers to the planning and continuous refining of the contents of the immediately upcoming releases
- The roadmap should be thought of as is a *view into the product backlog* that depicts how a particular solution (or line of solutions) is currently planned to evolve in the foreseeable future; roadmaps as separate documents get obsolete quickly
- A roadmap should display, preferably in visual form the release dates, the features planned to be included along with their estimated sizes, services that demand the developers' attention and the planned resource usage
- A common conceptual view of the product and its major features may be lacking even when the development organization is small
- Even in small organizations, possible long-term plans and strategic ambitions the top management has may not be visible if attention is not explicitly devoted to communicating these
- In the light of the philosophy of just-in-time progressive requirements elaboration prescribed by agile methods (see section 2.2.9 on pages 29-31), the practical applicability of the methods and techniques for release planning and requirements prioritization presented in software engineering literature is questionable
- The servicing needs of already launched products may be considerable, and a failure to account for this in long-term planning jeopardizes sustainability

5.5.3 Small companies need explicit portfolio management

RQ3: Do small software organizations suffer from the lack of explicit portfolio management?

- Small software organizations are often dealing with a portfolio of many different kinds of project and non-project activities instead of a clear-cut project or product development portfolio
- As only some of the activities that require the developers' attention are conducted as explicit projects or even concern the products the organization offers, the terms 'project portfolio management' and 'product portfolio management' are misleading; this dissertation proposes the term *development portfolio* ("*tekemissalkku"* in Finnish)
- The lack of an explicit portfolio management process does not necessarily cause problems: the mix of ongoing activities may be simple enough to be managed project-wise, or even without formal project management
- The need for explicit portfolio management in small software organizations is increased by organization growth, the multiple and sometimes conflicting roles and responsibilities of the key personnel, and leveraging customer-specific projects for product development purposes
- The problems associated with an inadequate portfolio management process are excessive multitasking, firefighting, overload, ineffective decision-making, missing strategic alignment, slipping schedules, project failures and poor profitability – as well as the perceived need to improve project management
- Although the root causes of the problems may be complex, setting up explicit processes for portfolio level decision-making may still prove a reasonably effective medicine
- Having well-working practices for managing individual development activities is not a prerequisite for setting up portfolio management; on the other hand, well-functioning

portfolio management, whether implicit or explicit, may be a pre-requisite for the success of the individual development activities

5.5.4 Portfolio management for the small software organization

RQ4: How can portfolio management be set up and conducted in small software organizations striving for agile software development?

Key actions in setting up and performing development portfolio management in small software organizations are:

- Appointing a group of people to be responsible for portfolio-level decision-making
- Building a publicly visible list of all ongoing activities that require time from development, including the information on who are assigned to which activity
- Synchronizing the portfolio
- Meeting regularly at portfolio synch-points to keep the list of ongoing activities up-to-date, perform short-term prioritization and setting the default resource allocation until the next meeting
- Agreeing on how decisions affecting more than one ongoing activity are made in urgent, 'emergency' type situations
- Identifying the different types of development activities, setting target spending levels that reflect the organization's strategy, and possibly tracking the actual spending
- Setting a limit to the number of concurrent activities a person can be involved in to curb excessive multi-tasking
- Tracking the developers' or development teams' workload
- Ensuring that performance metrics and incentives do not steer people towards local optimization

5.5.5 Linking business priorities with daily tasks

RQ5: How can long-term product and business planning be linked to daily software development tasks in agile software development? What role does portfolio management play here?

- To link business priorities with daily work, the trace of how the "smaller", iteration level work items have been split from the "larger", business and roadmap level goals should be kept
- Instead of a single portfolio management process, lean/agile software development involves multiple levels of portfolio decision-making
- *Product portfolio management* is concerned with deciding on the set of products and services offered and developed by the organization
- *Development portfolio management* deals with tactical resource allocation and prioritization across the set of possible activities that compete for the same pool of resources.
- When developers are involved in more than one concurrent activity, their *daily work* involves choosing from a portfolio of tasks the next one to work on
- If the activities are planned employing a hierarchical system of goals the business priorities become more transparent
- When decisions can be taken nearer to the level where the actual work is being done, fewer escalations are needed, decision-making gets faster and the quality of the decisions improves
- Although agile software development does not emphasise tools, maintaining hierarchical systems of goals for the entire development portfolio may make require tool support

5.6 Directions for future research

Both the results as well as identified limitations of this thesis yield a number of directions that

should be further explored. These are discussed below. Sections 5.6.1-5.6.4 focus on tackling the limitations of this study, while sections 5.6.5-5.6.8 discuss new avenues opened up by our results.

5.6.1 Examining the case organizations' representativeness

From the perspective of the practical implications of this study, gaining a better understanding of case organizations' representativeness is the most crucial aspect of future work. Several specific questions arise above the others

First, to what degree do the case organizations represent the total population of small organizations that develop software? Is it common that small organizations have – like our case organizations – relatively complex development portfolios composed of both product development and providing related (as well as non-product development related) services? Would the problems be better alleviated by practicing agile "by-the-book"? And to what degree do those companies who claim to practice agile "by the book" actually do so? Related to this, there is also the question of whether there in some situations are valid and practical reasons for assigning people or teams to more than one concurrent activity. What part of the population is striving to adopt agile software development, and why? Are there some common activities that require the developers' attention but are usually left outside of the scope of using agile methods? To what degree do the results from this dissertation apply to companies that, for one reason or another, do not strive to adopt agile software development? This is especially interesting, as it is possible to argue that iterations are crucial in any systems development project regardless of methodology (Berente & Lyytinen 2007).

Second, from the perspective of the results' applicability, are there key differences between small companies and small, relatively independent software development organizations of larger companies? What situational factors (Bekkers, Weerd & Brinkkemper 2008) may cause things to be different for small software companies and small independent software development organizations that are part of a larger company?

Third, what are the key differences of small and large software development organizations, and how do they affect the results' applicability? Does the presented framework and tool support scale, and if they do, what are the limits?

Lastly, from the perspective of future research efforts it would also be crucial to understand what kind of bias, if any, does the researchers' co-operative relationship with the organizations cause. Some of the above questions should in the future be examined by complementing our action research / case study based approach with surveys.

5.6.2 When is portfolio management an incorrect medicine?

Because of its fundamental role in moderating the flow of strategy to action, explicit portfolio management can help alleviate the problems and challenges experienced by an organization. However, it is likely that there are situations when at least some of the problems' root causes lie elsewhere than in inadequate portfolio management. While explicit portfolio management could – much in the way that is claimed of Scrum in (Schwaber 2007) – eventually force these root causes into the open to be dealt with, it is unknown how effective this is in solving actual problems. While explicit portfolio management may be effective on the long run, attending to some critical people (etc.) issues first might catalyze the improvement process to a significant degree.

We did not in the process of conducting the research for publication V encounter resistance in setting up and conducting explicit portfolio management in the case organizations. However, this may due to the fact that the companies were continuing co-operation with us with the expectation that explicit portfolio management will address their problems, or simply coincidental. Thus, while

there may be situations in which other issues will be have to be dealt with before setting up explicit portfolio management is realistic, this study does little to address the issue.

While publication IV does not discuss this in depth, we noticed that at Theseus, the management was, in relative terms, less satisfied of the current work practices than the development people. And, in the case organizations which did not initially practice explicit portfolio management, the management at first seemed in relative terms happier with the current work practices than the development. This left the author with the impression that managers' satisfaction to the current ways of working may actually be a tell-tale sign of a need to improve portfolio management. This should be explored further.

Further light on these issues could be gained by deliberately choosing case organizations whose management does not, at least from the outset, view portfolio management as something relevant to improve on. A situation where the views of the researchers and the organization's management would initially differ on the relevance of improving portfolio management would also provide possibilities to identify possible barriers to directly setting up explicit portfolio management.

5.6.3 How to pass further market tests

The "weak market test" is passed when a manager is willing to apply the construct to an actual decision-making problem (Kasanen, Lukka & Siitonen 1993). As explained in section 4.5.3, the ATMAN framework presented in section 4.5 and its current software implementation in Agilefant can be considered to have passed the weak market test. Because in a constructive case study it is practically impossible to go beyond the weak market test (Lindholm 2008), a natural next step would thus be to understand how to examine whether semi-strong and strong market tests could be passed.

The "semi-strong market test" is passed if the construct is widely adopted by companies (Kasanen, Lukka & Siitonen 1993). For this, we would need to define in an acceptable way what 'widely' as well as 'adopted' mean in practice. For example, it is fairly comfortable to state that Scrum is widely in use in the population of software organizations that practice agile software development (Kniberg & Skarin 2010). But, the level of adoption may differ – while many organizations strive to practice Scrum by-the-book, it is plausible that there are even more organizations who are only in the beginning steps of Scrum adoption. And, when not pressed for further evidence, what the representatives of an organization report on their level of adoption may or may not be reliable due to various reasons (Hansson et al. 2006).

While there based on the 2009 Agilefant user survey (see section 4.5.3) seems to be a growing community of Agilefant users, the vast majority of the organizations in question most likely have not currently adopted using hierarchical systems of goals to manage their development activities. This is simply because the capabilities for this have not been available until up to quite recently. On the other hand, to qualify as an 'adopter' of the framework itself, an organization does not need to use Agilefant. Using a backlog management tool (which may or may not be software) that employs a suitably similar functionality in terms of breaking larger work items into smaller may qualify as well. Without prematurely going into details regarding the research methods, the author believes that it could be possible to collect sufficient evidence for passing the semi-strong market test with, for example, a carefully designed survey.

According to (Kasanen, Lukka & Siitonen 1993), passing the "strong market test" requires that the organizations applying the construct systemically produce better results (measured in economic terms) than those not using it. Because of the nature of software development, evaluating this is considerably harder than in the previously discussed case. For example, while Scrum (Schwaber

2009) could perhaps be argued to have been widely adopted in industry, there may not currently be enough evidence to support whether even Scrum would pass the strong market test or not. Likewise, coming up with feasible ways to research methods-wise evaluate this is harder as well. In accordance with the agile principles of just-in-time elaboration (Poppendieck and Poppendieck 2009), discussing this further is out of the scope of this dissertation. The topic will be returned to if and when a version of our framework has passed the semi-strong market test.

5.6.4 A generic roadmap visualization for agile SW development

Based on our review of existing literature, there's plenty of research space to cover both regarding roadmapping as well as release planning. These are discussed below as well as and in section 5.6.6.

Publication III presents an example of a roadmap visualisation used by a single organization, but does not address its applicability to other organizations or other situations. Also, it could be argued that the visualization is still rather complex, and a simpler one would be easier to adopt and apply.

As proposed in this dissertation, in an ideal situation a visual roadmap could be directly generated or otherwise discerned from the product backlog itself. However, what information should such roadmap contain, and how should this vary according to the situation at hand? A way forward would be to collect real 'live' roadmaps that are actually used by organizations developing software, abstracting a set of 'basic roadmap elements', categorizing the roadmaps based on their content, and seeing in what kinds of situations are certain kinds of roadmaps appropriate.

5.6.5 Improved illustration for the "product backlog" in agile

Because agile methods' notion of using a product backlog to drive development efforts engulfs much of the entire area of software product management, it is one of the most complex issues in agile software development. It could even be argued that proper use of the product backlog is the heart of agile software development.

Thus, it is unfortunate, that existing illustrations and their explanations depict the concept and its subtleties quite poorly. In each of the illustrations and descriptions of the Product Backlog the author encountered when preparing section 2.2 (*Product management*), crucial aspects were missing. This is especially true for, although not limited, to the aspects related to long-term product and release planning (not to mention portfolio management). To include at least some the identified subtleties of backlog management, we felt compelled to put together a completely new illustration of "how the product backlog actually works" (see Figure 2.10 on page 36).

Thus, an important step in future research on agile software development would be to develop an improved, yet concise illustration and description (referred to as 'the construct' in the following) of what the product backlog is and how it should work, as well as to validate that the such an illustration actually is an improvement over the existing ones. We have envisioned that the validation of the construct could be performed via surveying software engineering professionals that are (to a varying degree) familiar with agile software development.

The first part of answering the survey would consist of selecting from a number (4-6) of descriptions the one that best fits the respondent's mental model of how the product backlog works. The second part of the survey would consist of selecting from a number of illustrations the one that best depicts the chosen description. During the second step, the respondent would be able to see all the descriptions as well as the illustrations (not matched to the descriptions) as well as to change his original choice from the first step. All respondents would be asked to finish the survey by writing a brief justification of why the selected description and illustration best matched his understanding of the concept. Also, selected respondents would be interviewed regarding their choice. The gained

feedback would be used to improve the construct for further rounds of the survey. Survey rounds, each time with new, "non-contaminated" participants would be continued until a significant number of respondents identify the construct as better describing the concept of agile requirements management using the product backlog than its existing alternatives.

5.6.6 Electronic versus physical backlog management support

As discussed in section *Implications for work item management tool support* (2.2.13), while there are proponents for both ways, current literature does not offer a conclusion on whether and/or when to use electronic or physical support "tools" for lean/agile backlog management. As the answer most likely is that in some cases an organization should prefer electronic tools, in others physical tools are appropriate, while still other cases might succeed via combining electronic and physical backlog management.

As it seems likely that using an electronic support tool is critical from the perspective of adopting and using hierarchical work item structures, the conditions for all the above cases should be explored further.

5.6.7 Requirements for backlog management tool support

According to our review of the literature (see section 2.2.13), there is currently very little practitioner and virtually no academic advice on what kind of functionality practitioners should look for when selecting support tools for backlog management.

As our results propose quite stringent demands from the perspective of requiring support for hierarchical work items, many of the existing tools on the market may simply be inadequate to apply beyond the single backlog / single product owner context. Thus, requirements for backlog management tool support in different situations should be explored further. It also seems plausible that organizations currently striving towards lean/agile software development have devised various ways to work around this problem. From a practitioner perspective, it would be useful if the more (and conversely, less) successful approaches were identified and evaluated.

5.6.8 Release planning in-the-large and algorithmic approaches

With respect to release planning, a first step forward would be to understand how the existing understanding in terms of algorithmic approaches to release content optimization could actually be used in the context of agile software development.

Also, to our knowledge, while agile software development is currently conducted in larger organizations where the single-team-single-product-owner-single-backlog precondition does not hold, there is currently very little research concerning the state of practice or how scaled-up agile release planning should actually be conducted (Heikkilä, Rautiainen & Jansen 2010). These questions should be tackled in the future.

5.7 Reflection on the research approach and methods

This study involved a rather significant amount of action research in the form of qualitative case studies, which served as input to the design science and constructive research approach taken (see section 3.2 *Research approach and case companies*). While such an approach is at best very capable for both making theoretical contributions as well as assisting in solving practitioners' problems, combining these is also an extremely challenging way of conducting research (Sein et al. 2011). However, the challenges may further be complicated because of the fact that much of

literature on design science ignores that in practice, the artifacts produced

> *…emerge from design, use, and ongoing refinement […] instead of "design and development" and "evaluation," where a suitable demonstration context is selected after building the artifact.* (Sein et al. 2011)

Sein et al (2011) argue that this has resulted in attempts to keep action research and design research conceptually apart, at least in terms of describing research methodology, even when the two are used as part of the same research process.

Reflecting on the above points, we recognize that this thesis, at least as far as the methodology description is concerned, may somewhat suffer from similar issues as well. For example, while the research questions and the problem formulation in reality were guided by the research process, and our results also evolved iteratively, this was not always explicit to us at the time the research was being conducted. The iterations and emphasis on evolving the research results, in other words, the (from a practitioner perspective) solution(s) that have been explicitly built into the action design research approach (Sein et al. 2011) could have speeded up the research process. This could also have helped in the converging of the results gained at seemingly different "fronts" into a single study, and thus, a dissertation.

Thus, we strongly recommend other researchers who are working in close, almost consultant-like relationship with industry partners to look into and familiarize themselves with recent methodological developments regarding the action design research approach.

6 References

Aguanno 2005, "Managing agile projects is different" in Managing Agile Projects, ed. K.J. Aguanno, Multi-Media Publications Inc., pp. 69-74.

Aken 2004, "Management research based on the paradigm of the design sciences: The quest for field-tested and grounded technological rules", Journal of management studies, vol. 41, no. 2, pp. 219-246.

Akker, Brinkkemper, Diepen & Versendaal 2008, "Software product release planning through optimization and what-if analysis", Information and software technology, vol. 50, no. 1-2, pp. 101-111.

Alajoutsijärvi, Mannermaa & Tikkanen 2000, "Customer relationships and the small software firm: A framework for understanding challenges faced in marketing", Information and management, vol. 37, no. 3, pp. 153-159.

Al-Emran, Pfahl & Ruhe 2010, "Decision support for product release planning based on robustness analysis", in Proceedings of the 18th international IEEE requirements engineering conference (RE'10), Sydney, Australia: IEEE Computer Society, pp. 157-166.

Ambler 2008a, "Beyond functional requirements on agile projects", Dr.dobb's journal, vol. 33, no. 10, pp. 64-66.

Ambler 2008b, "Complex requirements on an agile project", Dr.dobb's journal, vol. 33, no. 12, pp. 49-51.

Ambler 2008c, "Has agile peaked?", Dr.dobb's journal, vol. 33, no. 6, pp. 52-54.

Anderson 2010, Kanban, Sequim, Washington: Blue Hole Press.

Artto, Martinsuo, Gemnnden & Murtoaro 2009, "Foundations of program management: A bibliometric view", International journal of project management, vol. 27, no. 1, pp. 1-18.

Artz, Weerd, Brinkkemper & Fieggen 2010, "Productization: Transforming from developing customer-specific software to product software", in Proceedings of the first international conference on software business (ICSOB 2010), Jyväskylä, Finland: Springer, pp. 90-102.

Azar, Smith & Cordes 2007, "Value-oriented requirements prioritization in a small development organization", IEEE software, vol. 24, no. 1, pp. 32-37.

Baskerville & Myers 2004, "Foreword to special issue on action research in information systems: Making IS research relevant to practice", MIS quarterly, vol. 28, no. 3, pp. 329-335.

Beattie & Fleck 2005, "New perspectives on strategic technology management in small high-tech companies", in Proceedings of the 2005 IEEE international engineering management conference, St. John's, Newfoundland: IEEE, pp. 313-318.

Beck & Fowler 2001, Planning extreme programming, Canada: Addison Wesley.

Beck & Andres 2004, Extreme programming explained: Embrace change, 2nd ed., Boston, MA: Addison-Wesley.

Bekkers, Weerd & Brinkkemper 2008, "The influence of situational factors in software product management: An empirical study", in Second international workshop on software product management (IWSPM '08), IEEE, pp. 41-48.

Bekkers, Weerd, Spruit & Brinkkemper 2010, "A framework for process improvement in software product management", in Proceedings of the 17th european conference on systems, software and services process improvement (EuroSPI 2010), Grenoble, France: Springer, pp. 1-12.

Berander & Jönsson 2006, "Hierarchical cumulative voting (HCV) prioritization of requirements in hierarchies", International journal of software engineering and knowledge engineering, vol. 16, no. 6, pp. 819-849.

Berente & Lyytinen 2007, "What is being iterated? reflections on iteration in information system engineering processes" in Conceptual modelling in information systems engineering, eds. J. Krogstie, A.L. Opdahl & S. Brinkkemper, Springer, pp. 261-278.

Berry 1996, "Technical entrepreneurship, strategic awareness and corporate transformation in small high-tech firms", Technovation, vol. 16, no. 9, pp. 487-522.

Berry & Taggart 1998, "Combining technology and corporate strategy in small high tech firms", Research policy, vol. 26, no. 7-8, pp. 883-895.

Blichfeldt & Eskerod 2008, "Project portfolio management - there's more to it than what management enacts", International journal of project management, vol. 26, no. 4, pp. 357-365.

Boehm & Turner 2003, Balancing agility and discipline : A guide for the perplexed, Boston, MA: Addison-Wesley.

Brereton, Kitchenham, Budgen, Turner & Khalil 2007, "Lessons from applying the systematic literature review process within the software engineering domain", Journal of systems and software, vol. 80, no. 4, pp. 571-583.

Cao & Ramesh 2008, "Agile requirements engineering practices: An empirical study", IEEE software, vol. 25, no. 1, pp. 60-67.

Cardozo, Neto, Barza, Franca & Da Silva 2010, "SCRUM and productivity in software projects: A systematic literature review", in Proceedings of the 14th international conference on evaluation and assessment in software engineering (EASE), Keele University, UK: School of Computing and Mathematics at Keele University, UK, pp. 1-4.

Chaston 2009, Entrepreneurial management in small firms, Sage Publications.

Christensen & Raynor 2003, The innovator's solution: Creating and sustaining successful growth, Boston, MA: Harvard Business School Press.

Cockburn 2007, "What engineering has in common with manufacturing and why it matters", CrossTalk, vol. 20, no. 4, pp. 4-7.

Cohen, Lindvall & Costa 2005, "The roots of agility" in Managing Agile Projects, ed. K.J. Aguanno, Multi-Media Publications Inc., pp. 420.

Cohen 2010, Agile excellence for product managers: A guide to creating winning products with agile development teams, Super Star Press.

Cohn 2004, User stories applied: For agile software development, Addison-Wesley Professional.

Cohn 2009, Succeeding with agile: Software development using scrum, Upper Saddle River, NJ: Addison-Wesley.

Cohn 2005, Agile estimating and planning, Prentice Hall.

Condon 2002, Software product management - managing software development from idea to

product to marketing to sales, Boston, MA: Aspatore Books.

Cooper 2008, "Perspective: The stage-gate® idea-to-launch process - update, what's new, and NexGen systems", Journal of product innovation management, vol. 25, no. 3, pp. 213-232.

Cooper 2009, "How companies are reinventing their idea-to-launch methodologies", Research technology management, vol. 52, no. 2, pp. 47-57.

Cooper, Edgett & Kleinschmidt 2002, "Portfolio management: Fundamental to new product success" in The PDMA toolbook for new product Development , eds. P. Belliveau, A. Griffin & S. Somermeyer, New York, NY: John Wiley & Sons, Inc., pp. 331-364.

Coplien & Bjørnvig 2010, Lean architecture: For agile software development, Wiley.

Cruzes & Dybå 2011, "Research synthesis in software engineering: A tertiary study", Information and software technology, vol. 53, no. 5, pp. 440-455.

Cusumano 2008, "The changing software business: Moving from products to services", Computer, vol. 41, no. 1, pp. 20-27.

Cusumano 2004, The business of software - what every manager, programmer and entrepreneur must know to thrive and survive and good times and bad, New York, NY: The Free Press.

Cusumano 2003, "Finding your balance in the products and services debate", Communications of the ACM, vol. 46, no. 3, pp. 15-17.

Cusumano, Crandall, MacCormack & Kemerer 2009, "Critical decisions in software development: Updating the state of the practice", IEEE software, vol. 26, no. 5, pp. 84-87.

Davison, Martinsons & Kock 2004, "Principles of canonical action research", Information systems journal, vol. 14, no. 1, pp. 65-86.

DeGregorio 2000, "Technology management via a set of dynamically linked roadmaps", in Proceedings of the 2000 IEEE international engineering management society (EMS - 2000), Albuquerque, NM: IEEE, pp. 184-190.

Denyer, Tranfield & van Aken 2008, Developing design propositions through research synthesis.

Derby & Larsen 2006, Agile retrospectives: Making good teams great, 1 ed., USA: Pragmatic Bookshelf.

dos Santos & Travis 2009, "Action research use in software engineering: An initial survey", in IEEE Computer society, pp. 414-417.

Doz & Kosonen 2008, Fast strategy: How strategic agility will help you stay ahead of the game, Upper Saddle River, NJ: Prentice Hall.

Dybå & Dingsøyr 2008, "Empirical studies of agile software development: A systematic review", Information and software technology, vol. 50, no. 9-10, pp. 833-859.

Dzamashvili-Fogelström, Gorschek, Svahnberg & Olsson 2010, "The impact of agile principles on market-driven software product development", Journal of software maintenance and evolution: Research and practice, vol. 22, no. 1, pp. 53-80.

Ebert 2009, "Software product management", Crosstalk, the journal of defence software development, vol. 2009, no. January, pp. 15-19.

Eisenhardt & Brown 1998, "Time pacing: Competing in markets that won't stand still", Harvard

business review, vol. 1998, no. 3-4, pp. 59-69.

Elonen & Artto 2003, "Problems in managing internal development projects in multi-project environments", International journal of project management, vol. 21, no. 6, pp. 395-402.

Fayad, Laitinen & Ward 2000, "Software engineering in the small", Communications of the ACM, vol. 43, no. 4, pp. 115-118.

Ferrari 2008, Product management for software - simple processes for great results, Mondo Strategies Press.

Fitzgerald, Hartnett & Conboy 2006, "Customising agile methods to software practices at intel shannon", European journal of information systems (special section on business agility and diffusion of information technology), vol. 15, no. 2, pp. 200-213.

Fleury, Hunt, Spinola & Probert 2006, "Customizing the technology roadmapping technique for software companies", in Proceedings of the 2006 portland international conference on management of engineering and technology (PICMET '06), Istanbul, Turkey: IEEE, pp. 1526-1538.

Flint 2009, "A conceptual model of software engineering research approaches", in Australian software engineering conference, Gold Coast, Australia: IEEE Computer Society, pp. 229-236.

Fricker, Gorschek & Glinz 2008, "Goal-oriented requirements communication in new product development", in Second international workshop on software product management (IWSPM'08), Barcelona, Spain: IEEE Computer Society, pp. 27-34.

Galen 2009, SCRUM product ownership - balancing value from the inside out: Stories, ideas, lessons and practices for becoming a great product owner, RGCG, LLC.

Gersick 1994, "Pacing strategic change: The case of a new venture", Academy of management journal, vol. 37, no. 1, pp. 9-45.

Gilbert 2004, "Wearing two hats: Analyst-managers for small software projects", IT professional, vol. 6, no. 4, pp. 34-39.

Gill, Nelson & Spring 1996, "Seven steps to strategic new product development" in The PDMA handbook of new product development, ed. M.D. Rosenau, John Wiley & Sons, pp. 19-34.

Glass, Vessey & Ramesh 2002, "Research in software engineering: An analysis of the literature", Information & software technology, vol. 44, no. 8, pp. 491-506.

Glass 2009, "Making research more relevant while not diminishing its rigor", IEEE software, vol. 26, no. 2, pp. 95-96.

Gorschek, Garre, Larsson & Wohlin 2007a, "Industry evaluation of the requirements abstraction model", Requirements engineering, vol. 12, no. 3, pp. 163-190.

Gorschek, Svahnberg, Borg, Loconsole, Börstler, Sandahl & Eriksson 2007b, "A controlled empirical evaluation of a requirements abstraction model", Information and software technology, vol. 49, no. 7, pp. 790-805.

Gorschek & Wohlin 2006, "Requirements abstraction model", Requirements engineering, vol. 11, no. 1, pp. 79-101.

Goth 2009, "Agile tool market growing with the philosophy", IEEE software, vol. 26, no. 2, pp. 88-91.

Griffin & Hauser 1996, "Integrating R&D and marketing: A review and analysis of the literature",

Journal of product innovation management, vol. 13, no. 3, pp. 191-215.

Haapala 2010, Enhanced tool support for daily work management in agile software development, master's thesis, Oulu University, industrial engineering and management.

Hansson, Dittrich, Gustafsson & Zarnak 2006, "How agile are industrial software development practices?", Journal of systems and software, vol. 79, no. 9, pp. 1295-1311.

Hart 1998, Doing a literature review - releasing the social science research imagination, SAGE Publications.

Heikkilä, Rautiainen & Jansen 2010, "A revelatory case study on scaling agile release planning", in 36th EUROMICRO conference on software engineering and advanced applications (SEAA 2010), Lille, France: IEEE, pp. 289-296.

Helferich, Schmid & Herzwurm 2006, "Reconciling marketed and engineered software product lines", in 10th international software product line conference (SPLC'06), 2006, Baltimore, Maryland: IEEE Computer Society Press, pp. 23-27.

Hitt, Ireland & Hoskisson 1997, Strategic management: Competitiveness and globalization: Concepts, West Publishing Company.

Hodgkins & Hohmann 2007, "Agile program management: Lessons learned from the VeriSign managed security services team", in Proceedings of the AGILE 2007 conference (AGILE 2007), Washington, DC: IEEE computer society, pp. 194-199.

Hofer 2002, "Software development in austria: Results of an empirical study among small and very small enterprises", in 28th EUROMICRO conference on software engineering and advanced applications, pp. 361-366.

Hoffman, Parejo, Bessant & Perren 1998, "Small firms, R&D, technology and innovation in the UK: A literature review", Technovation, vol. 18, no. 1, pp. 39-55.

Horvat, Rozman & Györkös 2000, "Managing the complexity of SPI in small companies", Software process: Improvement and practice, vol. 5, no. 1, pp. 45-54.

Hughes & Cotterell 2002, Software project management, Berkshire, England, UK: McGraw-Hill.

Ivarsson & Gorschek 2009, "Technology transfer decision support in requirements engineering: A systematic review of REj", Requirements engineering, vol. 14, no. 3, pp. 155-175.

Järvinen 2007, "Action research is similar to design science", Quality and quantity, vol. 41, no. 1, pp. 37-54.

Jennings & Beaver 1996, "The performance and competitive advantage of small firms: A management perspective", International small business journal, vol. 15, no. 2.

Kahn, Castellion & Griffin (eds) 2005, The PDMA handbook of new product development, 2nd ed., Hoboken, NJ: John Wiley & Sons.

Kalliney 2009, "Transitioning from agile development to enterprise product management agility", in Agile conference, 2009. AGILE '09. Chicago, IL: IEEE, pp. 209-213.

Karlsson, Dahlstedt, Regnell, Natt och Dag & Persson 2007, "Requirements engineering challenges in market-driven software development - an interview study with practitioners", Information and software technology, vol. 49, no. 6, pp. 588-604.

Karlström & Runeson 2005, "Combining agile methods with stage-gate project management", IEEE

software, vol. 22, no. 3, pp. 43-49.

Karlström & Runeson 2006, "Integrating agile software development into stage-gate managed product development", Empirical software engineering, vol. 11, no. 2, pp. 203-225.

Kasanen, Lukka & Siitonen 1993, "The constructive approach in management accounting research", Journal of management accounting research, vol. 5, no. Fall, pp. 243-264.

Keith 2010, Agile game development with scrum, Addison-Wesley Professional.

Kekäle 2001, "Construction and triangulation: Weaponry for attempts to create and test theory", Management decision, vol. 39, no. 7, pp. 556-563.

Kennaley 2010, SDLC 3.0: Beyond a tacit understanding of agile, Fourth Medium Press.

Kettunen 2009, "Adopting key lessons from agile manufacturing to agile software product development—A comparative study", Technovation, vol. 29, no. 6-7, pp. 408-422.

Kettunen 2007, "Extending software project agility with new product development enterprise agility", Software process improvement and practice, vol. 12, no. 6, pp. 541-548.

Kettunen & Laanti 2006, "Combining agile software projects and large-scale organizational agility", Software process improvement and practice, vol. 13, no. 2, pp. 183-193.

Kitchenham, Pearl, Budgen, Turner, Bailey & Linkman 2009, "Systematic literature reviews in software engineering - A systematic literature review", Information and software technology, vol. 51, no. 1, pp. 7-15.

Kittlaus & Clough 2009, Software product management and pricing: Key success factors for software organizations, Berlin, Germany: Springer-Verlag.

Kniberg & Skarin 2010, Kanban and scrum - making the most of both, lulu.com.

Koivisto & Rönkkö 2010, "Entrepreneurial challenges in a software industry", in First international conference on software business (ICSOB 2010), Jyväskylä, Finland: Springer, pp. 169-174.

Koivisto 2010, "How can academic business research support the finnish software industry?", in Proceedings of the first international conference on software business (ICSOB 2010), Jyväskylä, Finland: Springer, pp. 211-216.

Kostoff & Schaller 2001, "Science and technology roadmaps", IEEE transactions on engineering management, vol. 48, no. 2, pp. 132-143.

Krebs 2008, Agile portfolio management, Redmond, WA: Microsoft Press.

Krishnan & Ulrich 2001, "Product development decisions: A review of the literature", Journal of management science, vol. 47, no. 1, pp. 1-21.

Ktata & Levesque 2009, "Agile development: Issues and avenues requiring a substantial enhancement of the business perspective in large projects", in Proceedings of the 2nd canadian conference on computer science and software engineering (C3S2E '09), Montreal, Quebec, Canada: ACM, pp. 59-66.

Laanti 2008, "Implementing program model with agile principles in a large software development organization", in IEEE, pp. 1383-1391.

Lago, Muccini & Vliet 2009, "A scoped approach to traceability management", Journal of systems and software, vol. 82, no. 1, pp. 168-182.

Laitinen, Fayad & Ward 2000, "The problem with scalability", Communications of the ACM, vol. 43, no. 9, pp. 115-118.

Lamsweerde 2004, "Goal-oriented requirements engineering: A roundtrip from research to practice", in Keynote paper, requirements engineering conference, 2004. proceedings. 12th IEEE international, pp. 4-7.

Lamsweerde 2001, "Goal-oriented requirements engineering: A guided tour", in 5th IEEE international symposium on requirements engineering, Toronto, Ontario: pp. 249-261.

Larman & Vodde 2010, Practices for scaling lean & agile development: Large, multisite, and offshore product development with large-scale scrum, Upper Saddle River, NJ: Addison-Wesley Professional.

Larman & Vodde 2008, Scaling lean & agile development: Thinking and organizational tools for large-scale scrum, Westford, MA: Addison-Wesley Professional.

Lawrence & Yslas 2006, "Three-way cultural change: Introducing agile within two non-agile companies and a non-agile methodology", in Agile conference 2006, Minneapolis, MN: IEEE, pp. 255-259.

Lawson, Longhurst & Ivey 2006, "The application of a new research and development project selection model in SMEs", Technovation, vol. 26, no. 2, pp. 242-250.

Leffingwell 2011, Agile requirements: Lean requirements practices for software teams, programs and the enterprise, Addison-Wesley Professional.

Leffingwell 2009, The big picture of enterprise agility (rev. 2), http://scalingsoftwareagility.files.wordpress.com/2007/03/the-big-picture-of-enteprise-agilitywhitepaper.pdf.

Leffingwell 2007, Scaling software agility: Best practices for large enterprises, Reading, MA: Addison-Wesley Professional.

Leffingwell & Aalto 2009, A lean and scalable requirements information model for the agile enterprise, http://scalingsoftwareagility.files.wordpress.com/2007/03/the-big-picture-of-enteprise-agilitywhitepaper.pdf.

Lehto 2010, Using backlogs for linking long-term product plans and development tasks in agile software development, master's thesis, Aalto University, Software Business and Engineering Institute (SoberIT).

Lehto & Rautiainen 2009, "Software development governance challenges of a middle-sized company in agile transition", in Proceedings of the 2009 ICSE workshop on software development governance (SDG '09), Vancouver, Canada: IEEE computer society, pp. 36-39.

Lehtola 2006, Providing value by prioritizing requirements throughout software product development - state of practice and suitability of prioritization methods, licenciate's thesis, Helsinki University of Technology, Software Business and Engineering Institute (SoberIT).

Lehtola & Kauppinen 2006, "Suitability of requirements prioritization methods for market-driven software product Development ", Software process improvement and practice, vol. 11, no. 1, pp. 7-19.

Lehtola, Kauppinen, Komssi & Vähäniitty 2009, "Linking business and requirements engineering: Is solution planning a missing activity in software product companies?", Requirements engineering, vol. 14, no. 2, pp. 113-128.

Lehtola, Kauppinen & Kujala 2005, "Linking the business view to requirements engineering: Long-term product planning by roadmapping", in Proceedings of the 13th IEEE international requirements engineering conference (RE'05), Paris, France: IEEE, pp. 439-443.

Lindholm 2008, "A constructive study on creating core business relevant CREM strategy and performance measures", Facilities, vol. 26, no. 7/8, pp. 343-358.

Lukka & Kasanen 1995, "The problem of generalizability: Anecdotes and evidence in accounting research", Accounting, auditing & accountability, vol. 8, no. 5, pp. 71-90.

MacCormack 2001, "Product-development practices that work: How internet companies build software", MIT sloan management review, vol. 42, no. 2, pp. 75.

Martinsuo 2001, "Project portfolio management: Contingencies, implementation and strategic renewal" in Project portfolio management - strategic management through projects, eds. K. Artto, M. Martinsuo & T. Aalto, Project Management Association Finland, pp. 61-77.

Mc Elroy & Ruhe 2010, "When-to-release decisions for features with time-dependent value functions", Requirements engineering, vol. 15, no. 3, pp. 337-358.

Miettinen, Mazhelis & Luoma 2010, "Managerial growth challenges in small software firms: A multiple-case study of growth-oriented enterprises", in Proceedings of the first international conference on software business (ICSOB 2010), Jyväskylä, Finland: Springer, pp. 26-37.

Miles & Huberman 1994, Qualitative data analysis: An expanded sourcebook, SAGE Publications Inc.

Mishra & Mishra 2009, "Market-driven software project through agility: Requirements engineering perspective", in Springer Berlin Heidelberg, pp. 103-112.

Moe, Dingsyr & Kvangardsnes 2009, "Understanding shared leadership in agile development: A case study", in 42nd hawaii international conference on system sciences (HICSS '09), Big Island, HI: IEEE, pp. 1-10.

Mohan, Ramesh & Sugumaran 2010, "Integrating software product line engineering and agile development methods", IEEE software, vol. 23, no. 3, pp. 48-55.

Murphy & Gorchels 1996, "How to improve product management effectiveness", Industrial marketing management, vol. 25, no. 1, pp. 47-58.

Nambisan 2001, "Why service businesses are not product businesses", MIT sloan management review, vol. 42, no. 4, pp. 72-80.

Nambisan & Wilemon 2000, "Software development and new product development: Potentials for cross-domain knowledge sharing", IEEE transactions on engineering management, vol. 47, no. 2, pp. 211-220.

Ngo-The & Ruhe 2009, "Optimized resource allocation for software release planning", IEEE transactions on software engineering, vol. 35, no. 1, pp. 109-123.

Norja 2010, Designing a framework for linking company goals with daily tasks in a small software company, Aalto University, School of Science and Technology, Faculty of Information and Natural Sciences, Software Business and Engineering Institute (SoberIT).

Nuseibeh & Easterbrook 2000, "Requirements engineering: A roadmap", in Conference on the future of software engineering, Limerick, Ireland: ACM, pp. 35-46.

O'Connor 2004, "Spiral-up implementation of NPD portfolio and pipeline management" in The PDMA toolbook for new product Development , eds. P. Belliveau, A. Griffin & S. Somermeyer, John Wiley & Sons, pp. 461-492.

Oliveira & Rozenfeld 2009, "Integration of technology roadmapping and project portfolio management to improve the front-end of NPD process", in Portland international conference on management of engineering and technology (PICMET) 2009, Portland, OR: IEEE, pp. 2080-2089.

Paetsch, Eberlein & Maurer 2003, "Requirements engineering and agile software development", in Enabling technologies: Infrastructure for collaborative enterprises, 2003, Linz, Austria: IEEE, pp. 308-314.

Patton 2002, Qualitative research & evaluation methods, 3rd ed., Thousand Oaks, CA: Sage Publications.

Peffers, Tuunanen, Rothenberger & Chatterjee 2007, "A design science research methodology for information systems research", Journal of management information systems, vol. 24, no. 3, pp. 45-77.

Perry 2009, Business driven PMO setup: Practical insights, techniques and case examples for ensuring success, J. Ross Publishing.

Petersen & Wohlin 2010, "The effect of moving from a plan-driven to an incremental software development approach with agile practices", Empirical software engineering, vol. 15, no. 6, pp. 654-693.

Petticrew & Roberts 2005, Systematic reviews in the social sciences: A practical guide, Wiley-Blackwell.

Phaal, Farrukh, Mills & Probert 2003, "Customizing the technology roadmapping approach", in pp. 361-369.

Pichler 2010, Agile product management with scrum: Creating products that customers love, Addison-Wesley Professional.

Pino, Garcia & Piattini 2008, "Software process improvement in small and medium software enterprises: A systematic review", Software quality control, vol. 16, no. 2, pp. 237-261.

Pino, Pardo, García & Piattini 2010a, "Assessment methodology for software process improvement in small organizations", Information and software technology, vol. 52, no. 10, pp. 1044-1061.

Pino, Pedreira, García, Luaces & Piattini 2010b, "Using scrum to guide the execution of software process improvement in small organizations", Journal of systems and software, vol. 83, no. 10, pp. 1662-1677.

Poppendieck & Poppendieck 2009, Leading lean software development: Results are not the point, Addison-Wesley.

Potts 1993, "Software-engineering resarch revisited", IEEE software, vol. 10, no. 5, pp. 19-28.

Pressman 2010, Software engineering - a practitioner's approach, 7th ed., McGraw-Hill.

Pries & Quigley 2010, Scrum project management, CRC Press.

Racheva, Daneva & Buglione 2008, "Supporting the dynamic reprioritization of requirements in agile development of software products", in 2nd international workshop on software product management, Barcelona: IEEE.

Rajala, Rossi & Tuunainen 2003, "A framework for analysing software business models", in Proceedings of the 11th European Conference on Information Systems, New Paradigms in Organizations, Markets and Society. pp. 1-15.

Ramesh & Jarke 2001, "Toward reference models for requirements traceability", IEEE transactions on software engineering, vol. 27, no. 1, pp. 58-93.

Rautiainen 2004, Cycles of control: A temporal pacing framework for software product development management, licentiate's thesis, Helsinki University of Technology.

Rautiainen, Lassenius & Sulonen 2002, "4CC: A framework for managing software product development", EMJ - engineering management journal, vol. 14, no. 2, pp. 27-32.

Rautiainen, Lassenius, Vähäniitty, Vanhanen & Pyhäjärvi 2002, "A tentative framework for managing software product development in small companies", in Hawaii International Conference on System Sciences (HICSS-35), pp. 251-260.

Rautiainen, von Schantz & Vähäniitty 2011, "Supporting scaling agile with portfolio management: Case paf.com", in 44th hawaii international conference on system sciences (HICSS), pp. 1-10.

Rautiainen, Vuornos & Lassenius 2003, "An experience in combining flexibility and control in a small company's software product development process", in International symposium on empirical software engineering (ISESE 2003), Rome, Italy: IEEE Computer Society, pp. 28-37.

Regnell, Beremark & Eklundh 2000, "A market-driven requirements engineering process: Results from an industrial process improvement programme", Requirements engineering, vol. 3, no. 2, pp. 121-129.

Reinertsen 2009, The principles of product development flow - second generation lean product development, Redondo Beach, CA: Celeritas publishing.

Richardson & von Vangenheim 2007, "Why are small software organizations different?", IEEE software, vol. 24, pp. 18-22.

Richardson 2002, "SPI models: What characteristics are required for small software development companies?", Software quality control, vol. 10, no. 2, pp. 101-114.

Rico, Sayani & Sone 2009, The business value of agile software methods - maximizing ROI with just-in-time processes and documentation, J. Ross Publishing.

Rönkkö, Ylitalo, Peltonen, Koivisto, Mutanen, Autere, Valtakoski & Pentikäinen 2009, National software industry survey 2009, Helsinki University of Technology.

Rothman 2009, Manage your project portfolio: Increase your capacity and finish more projects, Raleigh, NC: Pragmatic Bookshelf.

Rothman 2007, Manage it! : Your guide to modern, pragmatic project management, Raleigh, NC: The Pragmatic Bookshelf.

Runeson & Höst 2009, "Guidelines for conducting and reporting case study research in software engineering", Empirical software engineering, vol. 14, no. 2, pp. 131-164.

Ruokonen 2008, "Market orientation and product strategies in small internationalising software companies", The journal of high technology management research, vol. 18, no. 2, pp. 143-156.

Saastamoinen & Tukiainen 2004, "Software process improvement in small and medium sized software enterprises in eastern finland: A state-of-the-practice study", in Proceedings of the 11th

european conference on software process improvement (EuroSPI 2004), Trondheim, Norway: Springer, pp. 69-78.

Schiel 2009, Enterprise-scale agile software development, Boca Raton: CRC Press.

Schwaber 2007, The enterprise and scrum, Redmond, WA: Microsoft Press.

Schwaber & Beedle 2002, Agile software development with scrum, Upper Saddle River, NJ: Prentice-Hall.

Schwaber & Sutherland 2010, Scrum guide, Agile Alliance, http://www.scrum.org/storage/scrumguides/Scrum%20Guide.pdf.

Sein, Henfridsson, Purao, Rossi & Lindgren 2011, "Action design research", MIS quarterly, vol. 35, no. 2.

Shalloway, Beaver & Trott 2009, Lean-agile software development: Achieving enterprise agility, Upper Saddle River, NJ: Addison-Wesley.

Sihvonen, Savolainen & Ahonen 2006, "The craving for external training in small and medium-sized software companies - A trigger effect towards software process improvement", in EuroSPI 2006, Potsdam, Germany: Springer,.

Sillitti & Succi 2005, "Requirements engineering for agile methods" in Engineering and managing software requirements, eds. A. Aurum & C. Wohlin, Springer Berlin Heidelberg, pp. 309-326.

Smith 2008, "Change: Embrace it, don't deny it", Research-technology management, vol. 51, no. 4, pp. 34-40.

Smits 2007, "The impact of scaling on planning activities in an agile software development center", in 40th annual hawaii international conference (HICSS2007), Waikoloa, HI: IEEE, pp. 274-281.

Sommerville 1996, Software engineering, Addison-Wesley Publishers Ltd.

Steindl 2005, "From agile software development to agile businesses", in 31st EUROMICRO conference on software engineering and advanced applications, http://www.computer.org/portal/web/csdl/doi/10.1109/EURMIC.2005.29: IEEE Computer Society Press, pp. 258-265.

Stober & Hansmann 2010, Agile software development - best practices for large software development projects, Berlin, Germany: Springer-Verlag.

Sutherland 2008, The first scrum: Was it scrum or lean?

Sutherland & Altman 2010, "Organizational transformation with scrum: How a venture capital group gets twice as much done with half the work", in Proceedings of the 43rd hawaii international conference on system sciences (HICSS-43), pp. 1-9.

Sutherland, Schoonheim & Mauritz 2009, "Fully distributed scrum: Replicating local productivity and quality with offshore teams", in Proceedings of the 42nd hawaii international conference on system sciences (HICSS-42), Big Island, HI: IEEE, pp. 1-8.

Sutherland, Viktorov, Blount & Puntikov 2007, "Distributed scrum: Agile project management with outsourced development teams", in 40th hawaii international conference on system sciences (HICSS-40), Big Island, HI: IEEE,.

Svahnberg, Gorschek, Feldt, Torkar, Saleem & Shafique 2010, "A systematic review on strategic release planning models", Information and software technology, vol. 52, no. 3, pp. 237-248.

Syed-Abdullah, Holcombe & Gheorge 2006, "The impact of an agile methodology on the well being of development teams", Empirical software engineering, vol. 11, no. 1, pp. 143-167.

Takeuchi & Nonaka 1986, "The new new product development game", Harvard business review, vol. 64, no. 1, pp. 137-146.

Taylor, Greer, Coleman, McDaid & Keenan 2008, "Preparing small software companies for tailored agile method adoption: Minimally intrusive risk assessment", Software process improvement and practice, vol. 13, no. 5, pp. 421-437.

Tengshe & Noble 2007, "Establishing the agile PMO: Managing variability across projects and portfolios", in Proceedings of the AGILE 2007 conference, Washington, DC: IEEE computer society, pp. 188-193.

Thomas & Baker 2008, "Establishing an agile portfolio to align IT investments with business needs", in Proceedings of the agile 2008, IEEE Computer Society, pp. 252-258.

Tikkanen, Kujala & Artto 2007, "The marketing strategy of a project-based firm: The four portfolios framework", Industrial marketing management, vol. 36, no. 2, pp. 194-205.

Treude & Storey 2010, "Awareness 2.0: Staying aware of projects, developers and tasks using dashboards and feeds", in 2nd ACM/IEEE international conference on software engineering (ICSE 2010), Cape Town, South Africa: ACM/IEEE, pp. 365-374.

Valkenhoef, Tervonen, Brock & Postmus 2010, "Product and release planning practices for extreme programming", in Proceedings of the 11th international conference on agile software development (XP2010), Trondheim, Norway: Springer, pp. 238-243.

Vanhanen, Itkonen & Sulonen 2003, "Improving the interface between business and product development using agile practices and the cycles of control framework", in Agile development conference 2003, Salt Lake City, Utah, USA: Agile Development Conference 2003, pp. 71-80.

Vidgen & Wang 2009, "Coevolving systems and the organization of agile software development", Info.sys.research, vol. 20, no. 3, pp. 355-376.

Vlaanderen, Jansen, Brinkkemper & Jaspers 2011, "The agile requirements refinery: Applying SCRUM principles to software product management", Information and software technology, vol. 53, no. 1, pp. 58-70.

Vlaanderen, Jansen, Brinkkemper & Jaspers 2009, "The agile requirements refinery: Applying SCRUM principles to software product management", in Proceedings of the 3rd international workshop on software product management (IWSPM 2009), Atlanta, Georgia: IEEE, pp. 1-10.

Vähäniitty 2006, Do small software companies need portfolio management, too? , Licentiate Thesis Helsinki University of Technology, Software Business and Engineering Institute (SoberIT) Available online at http://www.soberit.hut.fi/jvahanii/.

Vähäniitty 2004a, "Commercial product management" in Pacing software product development: A framework and practical implementation guidelines, eds. K. Rautiainen & C. Lassenius, Helsinki University of Technology, Software Business and Engineering Institute, pp. 19-35.

Vähäniitty 2004b, "Pipeline management" in Pacing software product development: A framework and practical implementation guidelines, eds. K. Rautiainen & C. Lassenius, Helsinki University of Technology, Software Business and Engineering Institute, pp. 37-48.

Vähäniitty 2003, Product strategy decisions in small software product businesses - a framework and experiences, Master's thesis, Helsinki University of Technology, Institute of Software Business and

Engineering (SoberIT).

Vähäniitty, Lassenius & Rautiainen 2002, "An approach to product roadmapping in small software product businesses", in Quality connection - 7th european conference on software quality, Helsinki, Finland: Center for Excellence Finland, pp. 12-13.

Wallin, Ekdahl & Larsson 2002, "Integrating business and software development models", IEEE software, vol. 19, no. 6, pp. 28-33.

Wangenheim, Weber, Hauck & Trentin 2006, "Experiences on establishing software processes in small companies", Information and software technology, vol. 48, no. 9, pp. 890-900.

Ward, Fayad & Laitinen 2001, "Thinking objectively: Software process improvement in the small", Communications of the ACM, vol. 44, no. 4, pp. 105-107.

Ward, Laitinen & Fayad 2000, "Management in the small", Communications of the ACM, vol. 43, no. 11, pp. 113-116.

Weerd, Brinkkemper, Nieuwenhuis, Versendaal & Biljsma 2006a, "On the creation of a reference framework for software product management: Validation and tool support", in International workshop on software product management (IWSPM 2006), Minneapolis, Minnesota, USA: IEEE, pp. 3-12.

Weerd, Brinkkemper, Nieuwenhuis, Versendaal & Biljsma 2006b, "Towards a reference framework for software product management", in Proceedings of the 14th IEEE international requirements engineering conference (RE'06), Minneapolis/St. Paul, MI: IEEE, pp. 319-322.

Wells, Phaal, Farrukh & Probert 2004, "Technology roadmapping for a service organization", Research technology management, vol. 47, no. 2, pp. 46-51.

Wilby 2009, "Roadmap transformation: From obstacle to catalyst", in IEEE Computer Society, pp. 229-234.

Woodward, Surdek & Ganis 2010, A practical guide to distributed scrum, 1 ed., IBM Press.

Yin 1994, Case study research: Design and methods, Thousand Oaks, CA: Sage Publications, Inc.

Ziemer & Calori 2007, "An experiment with a release planning method for web application development" in Software process improvement, eds. P. Abrahamsson, N. Baddoo, T. Margaria & R. Messnarz, Springer Berlin / Heidelberg, pp. 106-117.

APPENDIX A: GLOSSARY

Due to the conceptual and constructive nature of this study, we have presented many concepts from several disciplines, for example, from new product development, software engineering and agile software development. As we have seen in Chapter 2 (Related work), different authors and disciplines use different terms to relate to the same concepts – as well as the same terms to relate to different concepts.

To help the reader, this glossary lists the definitions of all of the key terms used in this dissertation summary and the included publications. The glossary lists only the "final definitions" of the terms. Thus, the definitions below may not apply in the beginning of chapter 2 (specifically, sections 2.2.1-2.2.5) – especially in the excerpts taken from related work. However, they do apply throughout the other sections and chapters.

Some terms are defined using other terms; in this case, we have typed the latter using *italic font.* Because of the varying definitions of terms in contemporary literature, we have had to redefine many commonly used terms to suit the purposes of this thesis. The definitions for terms starting with an asterisk (*) are the result of this study, and have been defined by us based on the work of other authors. We have included references where necessary.

As this glossary has been compiled after the closing of the literature review, we have taken the liberty of utilizing the *2011 Scrum Guide* by Ken Schwaber and Jeff Sutherland[1] for providing up-to-date the definitions for certain basic concept of Scrum whose definitions in earlier work were unclear or vague. These terms are denoted with a cross (†).

***Action(s)** – refers to people working on and getting *Tasks* done during *Iterations*. See Figures 4.7 and 4.8. Used in the adage 'From strategy to action and back again' of *ATMAN*.

***Activity** – the generic term to refer to any larger undertaking than a task that takes up time from people, for example a *project* or ongoing customer support. This is used throughout this dissertation to highlight the finding that even in project-based organizations much of the work that takes up time from people may not be explicitly organized as *projects*. See Figure 4.9.

Agile software development – An umbrella term for software development methodologies based on iterative development and where requirements and solutions evolve through collaboration between self-organizing cross-functional teams and the customer. See section 2.1.3.

***Agilefant** – the open source tool for managing *development portfolio* developed to validate and refine the conceptual work performed in our research

ATMAN – Acronym for "Approach and Tool support for development portfolio MANagement", a research project at Aalto University funded by our industrial partners and Tekes (the National Finnish Technology Agency). Part of the research for this thesis was conducted in ATMAN.

Backlog item – see *Work item*

Burndown (or burn-up) graph – A graphical representation of how much work (as a function of time) is estimated to remain in getting the work planned for an iteration or a release done

[1] http://www.scrum.org/storage/scrumguides/Scrum Guide - 2011.pdf

(Cohen 2010).

***Business** (with a capital initial) – refers to the *Top management* area in the *Cycles of Control* framework.

***Business area** – refers to a certain (set of) market(s) and the *product(s)* an *organization* operates in. Synonymous with the concept of *strategic investment theme* as defined by Dean Leffingwell (2011)

***Business goal** – An achievement in a *business area* that directly contributes to the *mission*. Expressed in non-software terms, as opposed to *epics*.

***Business management** – A cycle in the *Cycles of Control* framework responsible for setting the *business strategy* for a *business area,* and accepting the *product vision* formulated in *product management*. The *enterprise* and *business management* cycles merge when there is only a single *business area*.

***Business strategy** – The set of decisions made in *business management*. These include how to compete in the *business area* and formulating its *business goals* so that they form a coherent whole compatible with the current *investment level* set for the *business area*.

Company – see *enterprise*

Corporation – see *enterprise*

***Cycles of Control** (CoC) – A framework for analysing and describing the management of iterative and incremental software development. First introduced in a conference paper by Rautiainen et al. (2002), this thesis has refined the Cycles of Control framework to consist of seven cycles grouped into three areas: *top management* (*enterprise* and *business management* cycles), strategic release management (*product* and *development portfolio management* cycles) and software development management (*release*, *iteration* and *heartbeat* cycles)

***Daily work** – refers to the work between *heartbeats*, for example, working on *tasks*.

†**Daily Scrum** – A 15-minute time-boxed event for the *team* to synchronize their *actions* and discuss their plan for the next 24 hours. This is done by inspecting the work since the last daily scrum and forecasting the work that could be done before the next one. The daily scrum is an example of a *heartbeat* level practice.

***Development** (with a capital initial) – refers to the area *software development management* cycles in the *Cycles of Control* framework

***Development infrastructure** – One of the decisions of the *Organization* key decision area. Deals with the selection, acquiring and usage of development tools and environments and their sharing amongst activities (see paper II).

***Development model** (key decision area) – This decision area refers to how the product development process is structured in terms of cadence and control mechanisms to realise the intended *release strategy* (see paper II).

***Development people** – everyone who has something to do with software development; includes people with development and development management responsibilities

†**Development team** – the people who do the work of delivering a potentially releasable increment of "Done" *product* at the end of each *Sprint*.

***Development portfolio** ('tekemissalkku' in Finnish) – refers to the portfolio of current and immediately upcoming activities. Includes everything that takes up time and attention from the *development people*; for example customer-specific development, consulting, possible non-project work and so on, whether, whether or not these activities actually are conducted as *projects* or involve software development in the strict sense.

***Development portfolio management** ('tekemissalkun hallinta' in Finnish) – The cycle in the Cycles of Control framework that looks at the current and immediately upcoming development *activities* as a whole and is concerned with making appropriate resourcing decisions in a timely and organised manner. This cycle ultimately connects *Business* and *Development* decision-making, and is crucial for successful strategy implementation

Engineering – "the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people" (Merriam-Webster Online Dictionary)

***Enterprise** – an *organization* engaging in commercial activity to make profit by delivering value to its customers and the society. The term *company* is used synonymously with the concept of enterprise throughout this summary. In the included publications, the term *corporation* is also used.

***Enterprise management** – A cycle in the *Cycles of Control* framework responsible for setting the *enterprise strategy* and accepting the *business goals* formulated in *business management*.

***Enterprise strategy** – The set of decisions made in *enterprise management*. These encompass considering the overall direction of the *enterprise* in terms of its *business areas*, resource usage and general attitude towards growth. From the perspective of managing software development, enterprise strategy manifests in setting *investment levels* for its *business areas*.

***Epic** – A *user story* that is estimated to require too much effort to get done in a single *release*. Epics are "bold, impactful, marketable differentiators" (Leffingwell 2011) of an offering and contribute to the achievement of *business goals*.

***Excessive multitasking** – A problem associated with inadequate *development portfolio management*. In excessive multitasking, development people start working in a time-sharing manner in an attempt to show progress on all the different activities they have been assigned to. This, among other things, slows the completion of each activity (see paper IV).

Feature – A *user story* that is estimated as small enough to fit into a *release*, but too big to get done in a single iteration. Features are "short, descriptive, value delivery and benefit-oriented statement of system functionality" (Leffingwell 2011) and contribute to getting *epics* done.

***Firefighting** – refers to reactive and unplanned allocation of resources to solve and fix problems that are discovered late in a project or during maintenance (see paper IV).

Goal-oriented requirements engineering (GORE) is concerned with the use of goals for eliciting, elaborating, structuring, specifying, analyzing, negotiating, documenting, and modifying requirements (Lamsweerde 2001).

***Heartbeat** – The cycle in the *Cycles of Control* framework referring to the shortest time horizon on which work is monitored and synchronized. In heartbeats, the management focus is on *tasks* and coordinating the individual *actions* that lead up to the completion of *tasks*. The *Daily Scrums* of *Scrum* are an example of a heartbeat-level practice.

***Ineffective decision-making** – A problem associated with inadequate *development portfolio*

management. "Ineffective" is used here as an umbrella term for 1) late; 2) toothless (e.g., lacking clout); and 3) misguided and/or uninformed portfolio-level decision making. See paper IV for details.

***Investment levels** – A statement of intent regarding how much, in relative terms, of the total resources of the *enterprise* should be spent on its *business areas*.

Iteration – "One of several successive periods of time when all the work to complete one full slice of the finished product is performed; a project consists of multiple iterations, also referred to [in Scrum] as Sprints." (Schwaber 2007), p. 115; see also *Sprint*

***Iteration backlog** – The section of the *product backlog* containing the stories and the associated tasks a *team* is supposed to get done during the *iteration*. Referred to in Scrum as the *Sprint backlog*.

***Key decision areas** that connect *Business* and *Development* are *portfolio management*, *organisation*, *development model*, *product management* and *quality strategy*; see paper II.

***Key processes** that should connect business and development decision-making are *development portfolio management* (part of the *portfolio management* decision area), *roadmapping* and *release planning* (parts of the *product management* decision area); see paper II.

***Levels of planning** – According to the performed review of the literature, planning should in an organization striving for agile software development be performed on several levels. These are: *Strategy*, *Product portfolio*, *Product*, *Release*, *Iteration* and Day (see *Heartbeats* and *Daily work*). The higher levels set objectives while the lower levels add detail on how these objectives are to be achieved. See section 2.2.7 (*Levels of planning in agile software development*) for detail.

***Levels of portfolio management** – According to the performed review of the literature, agile software development implies instead of a single portfolio management process, each of the planning levels involves portfolio management decision-making. See section 2.3.5 (*Levels of portfolio management*) for detail.

Management – "conducting or supervising of something as a business" (Merriam-Webster Online Dictionary)

***Missing strategic alignment** – A problem associated with inadequate *development portfolio management*. Refers to a situation where the ongoing mix of activities has no apparent link to *enterprise strategy* or *business goals* (see paper IV).

Mission – A statement of the *organization's* unique purpose and the scope of its operations (Hitt, Ireland & Hoskisson 1997)

New product development The overall process of strategy, organization, concept generation, product and marketing plan creation and evaluation and commercialization of new products (Kahn, Castellion & Griffin 2005).

***Organization** (noun) – An organized group of people with a particular purpose (Oxford Dictionaries Online). Used throughout this dissertation summary to generalize from *enterprise* or *company*, or a part thereof.

***Organization** (key decision area) – One of the decision areas connecting *top management* and *software development*. Consists of deciding on Organisational design, Roles and

responsibilities, Team staffing, Team physical arrangement and location, Competences and Use of outsourcing (see paper II for further definitions).

***Overload** – A problem associated with inadequate *development portfolio management*. Overload occurs when too few people are trying to accomplish too much. A typical overload may be two to three times the actual capacity (see paper IV).

***Perceived need to "improve project management"** – A problem associated with inadequate *development portfolio management*. Inadequate *development portfolio management* may not be recognized as the root cause of the troubles experienced. Instead, the personnel may believe that better project management, e.g., more detailed planning or more precise effort estimates, would help (see paper IV).

Pipeline management – "integrates product strategy, project management and functional management to continually optimize the cross-project management of all development-related activities" (Kahn, Castellion & Griffin 2005). Thus, pipeline management is close to synonymous with *development portfolio management* as defined in this summary.

***Planning levels** – see *levels of planning.*

***Portfolio –** see *product portfolio.*

***Portfolio council** – The party responsible for *product portfolio management* and ultimately, *development portfolio management* decisions as well.

Portfolio management of new product development projects (a.k.a Project portfolio management) – the process for achieving balanced resource allocation in terms of value maximization, strategic alignment, risk level and the number of ongoing projects (Cooper, Edgett & Kleinschmidt 2002).

***Portfolio synch-point** – a meeting where the *portfolio council* keeps the list of ongoing *activities* up-to-date, performs short-term prioritization of the *development portfolio* and decides on the default resource allocation until the next portfolio synch-point.

***Product** – The generic term used for something the *organization* is developing either to be sold or for internal use. May be a piece of software and/or *service*.

Product backlog – "A prioritized list of functional and non-functional requirements and features to be developed for a new product or to be added to an existing *product*" (Schwaber 2007). See section 2.2.6 for details.

***Product management** – A cycle in the *Cycles of Control* framework responsible for setting the *product strategy.*

***Product strategy** – Refers to the set of decisions made in the *product management* cycle of the *Cycles of Control* framework. From the perspective of managing software development, this includes formulating the *product vision* and the *release strategy.* These are accepted by *business management*.

†Product owner – The product owner is responsible for maximizing the value of the *product* and the work of the *development team*. The product owner is the sole person responsible for managing the *product backlog*

***Product portfolio** – the set of *products* offered and being developed by an organization.

***Product portfolio management** – The decision-making process in *enterprise management*

used to set the relative levels of resource spending (a.k.a. *investment levels*) for its *business areas*.

Product roadmapping – see *roadmapping*

***Product vision** – A statement of the most desirable, future state of a *product* (Kahn, Castellion & Griffin 2005) explaining *the* essential value it will provide and for whom as well as "the business opportunities involved" (Woodward, Surdek & Ganis 2010)

Project – A non-routine undertaking that has specific objectives, a pre-determined time span and involves more than one person (Hughes and Cotterell 2002)

***Project failures and poor profitability** – A problem associated with inadequate *development portfolio management*. See paper IV.

***Progressive refinement of work items** – The notion of splitting large, vague work items over time into smaller, more clearly defined items in order to develop and deliver the most valuable functionality first. See section 2.2.8.

***Problems associated with inadequate portfolio managemement** – 1) *Excessive multitasking*, 2) *Firefighting*, 3) *Overload*, 4) *Ineffective decision-making*, 5) *Missing strategic alignment*, 6) *Slipping schedules*, 7) *Project failures and poor profitability* and 8) Perceived need to improve project management. See paper IV.

***Quality strategy** (key decision area) – This decision area consists of defining "good-enough" quality, risk management and test planning (see paper II).

***Release** – a software build that has been made available to one or more parties outside the *development team*.

***Release backlog** (also known as **Release plan**) – The section of the product backlog that contains those work items that are thought of as necessary to get done before the version of the product currently under work is launched. The release backlog evolves constantly, and is the result of the negotiation between *product management* and *development team(s),* moderated by *development portfolio management*.

***Release planning** – refers to the planning and continuous refining of the contents of the next *release*.

***Release project** – a *project* that aims at producing a *release*

***Release strategy** – Refers to the set release cycle and the plan of how the *product* should be developed in terms of high-level functionality and changes to the underlying technologies (that is, *features* and *epics*) and future resource needs. Communicated using the *roadmap.*

Requirements abstraction model – a *work item meta-model* by Gorsheck et al. (2006) that explicates both the *level of planning* as well as the requirements' level of abstraction; see section 2.2.9.

***Roadmap** – In the context of agile software development, the roadmap should be considered as a view into the *product backlog* that depicts how a particular *product* is currently planned to evolve in the foreseeable future. It shows the release dates, the planned *epics* and *features* and their estimated sizes, services that demand the developers' attention, and the planned resource usage.

***Roadmapping** – A common metaphor for planning the use of resources, technology and their

relationships over a period of time (Kostoff & Schaller 2001). In the context of agile software development, roadmapping refers to grooming the *product backlog* so that the *features* and *epics* for the foreseeable future (that is, upcoming releases beyond the one being currently worked on) are visible and reflect the current understanding of the relevant stakeholders.

***Root level (work) item** – A work item that has no parent items; see *work item hierarchy*.

Scrum – "A process for managing the development and deployment of complex products that is based on empirical process control theory and stands on the core practices of iterative development, which generates increments of product by using self-managing, cross-functional teams" (Schwaber 2007), p.115

Service – "A product which is (at least substantially) intangible. Servicing usually involves customer participation in some important way. Services cannot be sold in the sense of ownership transfer, and they have no title of ownership." (Kahn, Castellion & Griffin 2005) The concept of *product* as defined in this thesis includes the service concept.

***Slipping schedules** – A problem associated with inadequate development portfolio management. See paper IV.

Small company (or **Small organization**) – A company (or an organization) with less than 50 people

***Software development management** – Refers to the *release*, *iteration* and *heartbeat* cycles in the *Cycles of Control* framework.

Software engineering – deals with "processes, methods and tools that aim for enabling complex computer-based systems to be built in a timely manner with quality" (Pressman 2010)

Software product management – the process that governs a *product* from its inception to the market or customer delivery and service (Ebert 2009)

†**Sprint** – A time-box of one month or less during which a "Done", useable and potentially releasable product increment is created. The *Scrum* framework calls *iterations* Sprints.

Sprint backlog (a.k.a **sprint plan**) – The topmost section of the product backlog, containing contains those work items that have been committed to for the ongoing development iteration, including the tasks that are needed to get the work items done (Schwaber 2007)

***Story** – a *user story* estimated as small enough to be developed in a single *iteration*. Stories contribute to getting *features* done.

Strategic investment theme (a.k.a. **strategic product theme**, **investment theme**, or **theme** for short) – An initiative that drives the organization (Leffingwell 2011). Synonymous with the concept of *business area* as defined above.

***Strategic release management** – refers to the *product* and *development portfolio management* cycles in the *Cycles of Control* framework.

***Task** – Something relatively well-defined and effort-wise small that needs to get done. The *development team* defines the tasks that are needed to get the *stories* in the *sprint backlog* done as part of *sprint* planning.

Time pacing – The idea of dividing a fixed time period allotted to the achievement of a goal into fixed-length segments to help people synchronize the speed and intensity of their efforts. On a high level, "time pacing implies creating new products, launching new businesses, or

entering new markets according to the calendar"; adapted from (Eisenhardt & Brown 1998)

***Top management** – Refers to the *enterprise* and *business management* cycles of the *Cycles of Control* framework.

***Traffic control squad** – The party responsible for resolving mid-*iteration* conflicts and crises. See section 4.5.

***User story** – "A user story describes functionality that will be valuable to either a user or purchaser of a system or software. […] User stories represent customer requirements rather than document them." (Cohn 2004). *Epics*, *features* and *stories* are all user stories

***Work item** – any defined piece of work larger than a *Task*. Also referred to as *Backlog item*.

***Work item hierarchy** – The result of keeping track of *work item* parent-child relationships when practicing *progressive refinement of work items*.

***Work item meta-model** – A model for explicating the relative sizes of *work items* and the *planning levels* involved. For examples of work item meta-models, see Table 2 in section 2.2.9.

***Work queue** – A feature in *Agilefant* which allows an individual developer to choose and order those *tasks* that currently seem from his perspective the most relevant to attend to. The work queue can also be used for communicating the tasks accomplished since last *heartbeat* and the intended plan-of-action for the short term. See section 4.5.2.

APPENDIX B: DATA COLLECTION AND ANALYSIS TEMPLATES

To make the data collection and analysis done as part of action research leading up to the publication of the included papers more visible to the reader, this appendix contains selected templates used in data collection and/or analysis.

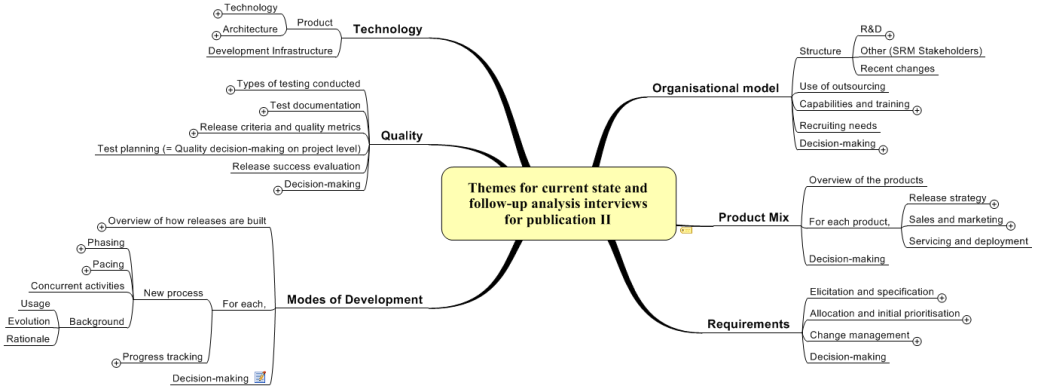


Figure B1 Interview themes and analysis framework for step 3) of answering RQ1

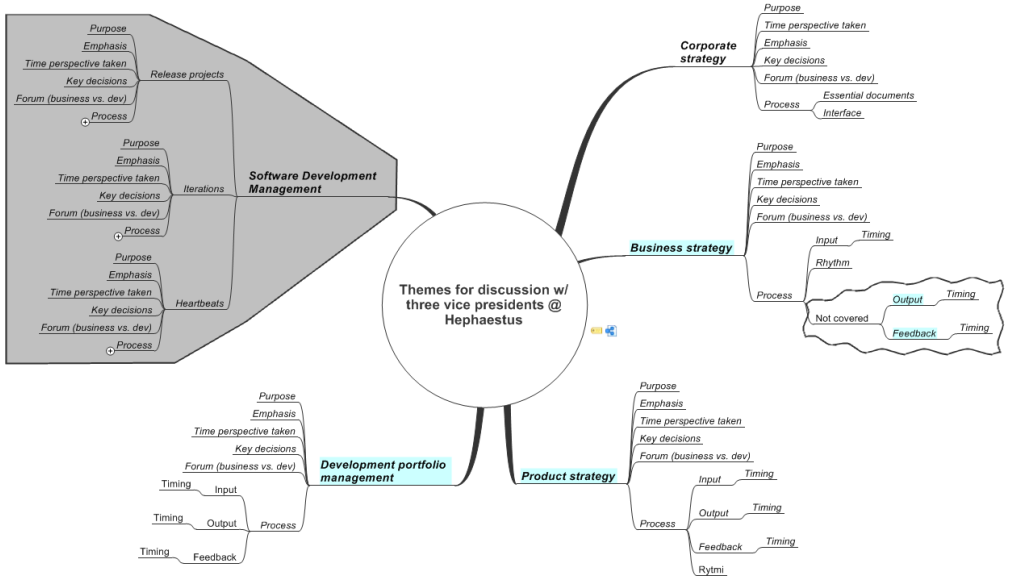


Figure B2 Interview themes and analysis framework for step 5) of answering RQ1

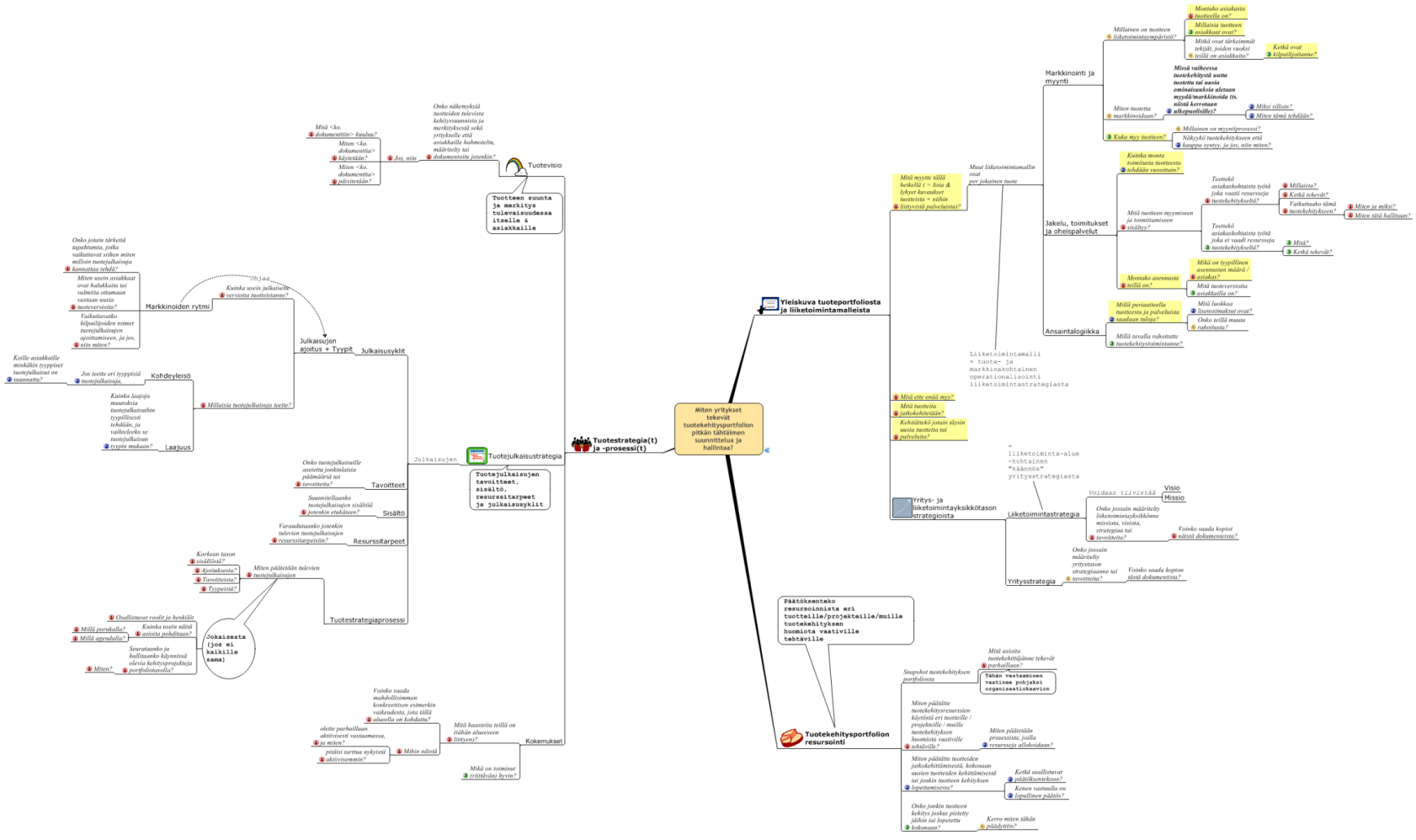


Figure B3 Initial state analysis interview themes and questions for step 1) of answering RQ3

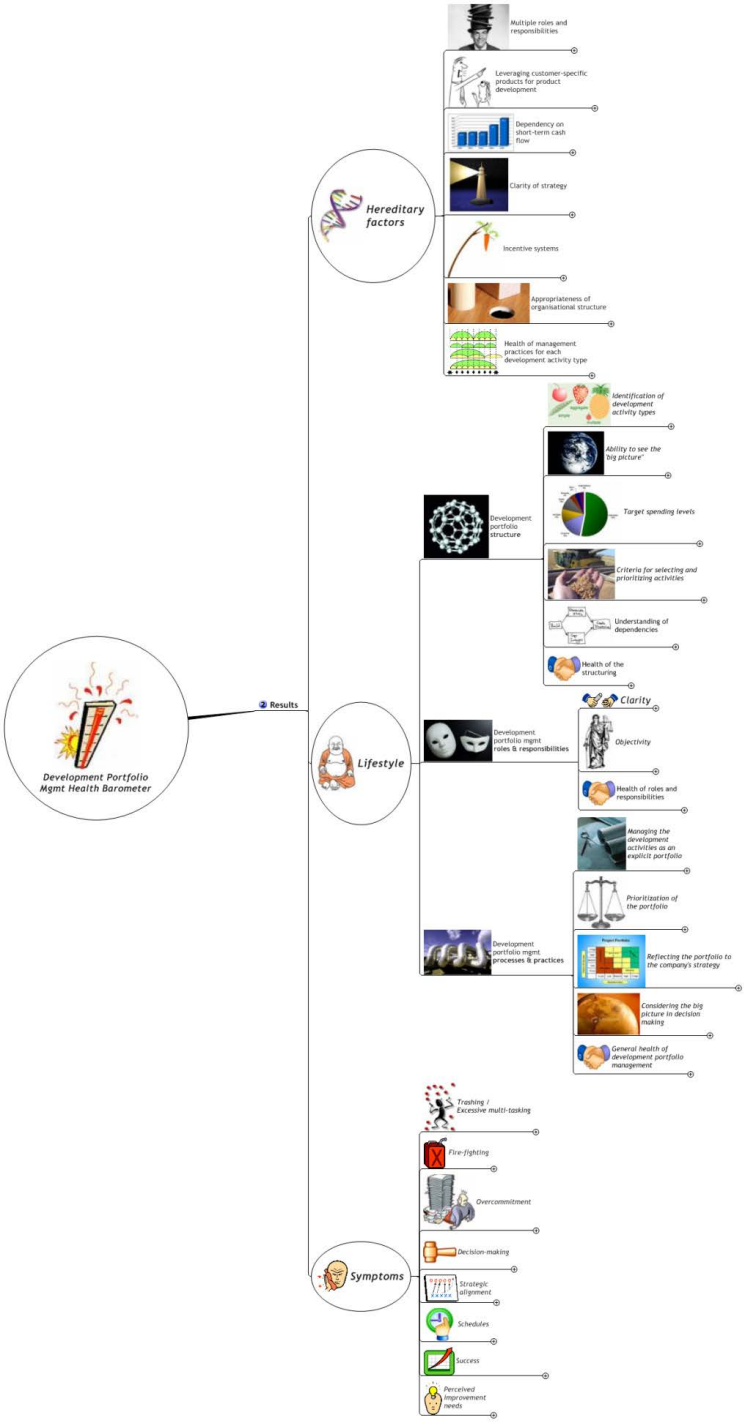
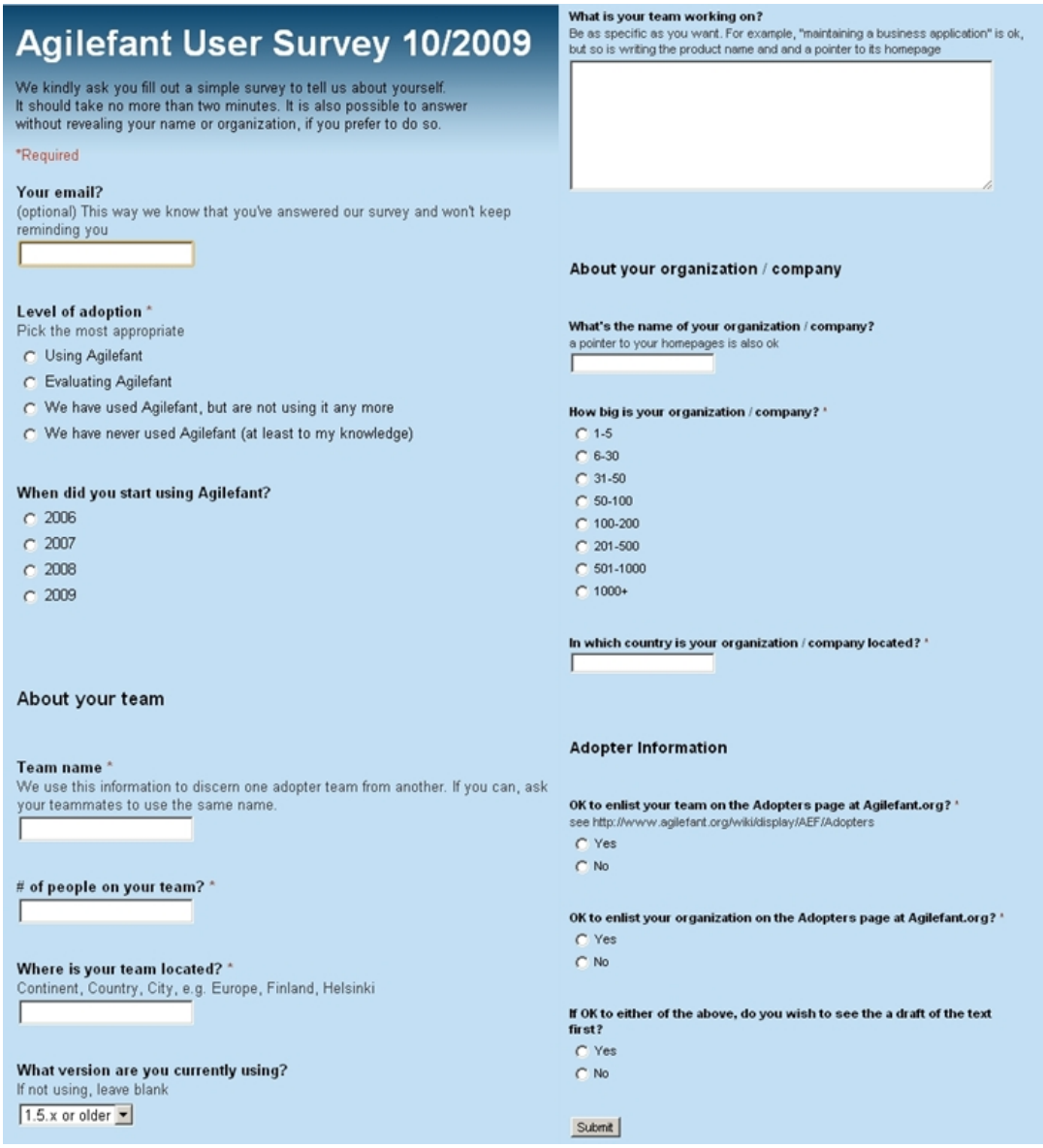


Figure B4 Interview themes and analysis framework for answering RQ4



Agilefant User Survey 10/2009

We kindly ask you fill out a simple survey to tell us about yourself.
It should take no more than two minutes. It is also possible to answer without revealing your name or organization, if you prefer to do so.

*Required

Your email?
(optional) This way we know that you've answered our survey and won't keep reminding you

Level of adoption *
Pick the most appropriate
- Using Agilefant
- Evaluating Agilefant
- We have used Agilefant, but are not using it any more
- We have never used Agilefant (at least to my knowledge)

When did you start using Agilefant?
- 2006
- 2007
- 2008
- 2009

About your team

Team name *
We use this information to discern one adopter team from another. If you can, ask your teammates to use the same name.

# of people on your team? *

Where is your team located? *
Continent, Country, City, e.g. Europe, Finland, Helsinki

What version are you currently using?
If not using, leave blank
1.5.x or older

What is your team working on?
Be as specific as you want. For example, "maintaining a business application" is ok, but so is writing the product name and and a pointer to its homepage

About your organization / company

What's the name of your organization / company?
a pointer to your homepages is also ok

How big is your organization / company? *
- 1-5
- 6-30
- 31-50
- 50-100
- 100-200
- 201-500
- 501-1000
- 1000+

In which country is your organization / company located? *

Adopter Information

OK to enlist your team on the Adopters page at Agilefant.org? *
see http://www.agilefant.org/wiki/display/AEF/Adopters
- Yes
- No

OK to enlist your organization on the Adopters page at Agilefant.org? *
- Yes
- No

If OK to either of the above, do you wish to see the a draft of the text first?
- Yes
- No

Submit

Figure B5 Agilefant user survey form used in step5) of answering RQ5

APPENDIX C: PUBLICATIONS

This appendix contains the six publications included in the dissertation.

I. Vähäniitty, Jarno. "A Tentative Framework for Connecting Long-Term Business and Product Planning with Iterative & Incremental Software Product Development". In proceedings of the 7th International Workshop on Economic-Driven Software Engineering Research (EDSER-7) at ICSE 2005, St. Louis, USA, 2005

II. Vähäniitty, Jarno. "Key Decisions in Strategic New Product Development for Small Software Product Businesses". In proceedings of Euromicro 2003. Antalya, Turkey, 2003

III. Vähäniitty, Lassenius, Rautiainen & Pekkanen: "Long-term Planning of Development Efforts by Roadmapping - a Model and Experiences from Small Software Companies". In proceedings of Euromicro 2009. Sep 2009, Patras, Greece

IV. Vähäniitty, Rautiainen & Lassenius: "Small Software Organizations Need Explicit Portfolio Management" IBM journal of Research and Development", Vol. 54, No. 2 (Issue on Creating Business Value through Software Development), April 2010

V. Vähäniitty, J & Rautiainen, K. "Towards an Approach for Development Portfolio Management in Small Product-Oriented Software Companies". In proceedings of Hawaii International Conference on System Sciences (HICSS-38), Jan 2005, Big Island, Hawaii, USA.

VI. Vähäniitty & Rautiainen: "Towards a Conceptual Framework and Tool Support for Linking Long-term Product and Business Planning with Agile Software Development". ICSE International Workshop on Software Development Governance (SDG2008), May 2008, Leipzig, Germany.

Most of the literature on agile software development has concentrated on the perspective of a single team in an individual development project, leaving the links to product and portfolio management largely unaddressed. Based on case studies of leading Finnish software organizations and a systematic review of scientific and practitioner literature, this dissertation describes how product and portfolio management can be understood in the context of agile software development. As proof-of-concept, we present Agilefant (www.agilefant.org), an open source tool for managing a portfolio of activities of which some – though not necessarily all – are planned and managed using backlogs. We propose that using hierarchical work item structures provides transparency to business priorities while still enabling just-in-time elaboration required by agile software development.

ISBN 978-952-60-4505-4
ISBN 978-952-60-4506-1 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934
ISSN 1799-4942 (pdf)

Aalto University
School of Science
Department of Computer Science and Engineering
www.aalto.fi

BUSINESS + ECONOMY

ART + DESIGN + ARCHITECTURE

SCIENCE + TECHNOLOGY

CROSSOVER

DOCTORAL DISSERTATIONS

Aalto-DD 15/2012

Jarno Vähäniitty

Towards Agile Product and Portfolio Management

Aalto University