Aalto University
School of Science
and Technology

Jarno Vähäniitty, Kristian Rautiainen
Ville Heikkilä & Kevin Vlaanderen

Towards Agile Product and Portfolio Management

A publication of the Software Process Research Group (SPRG),
Software Business and Engineering Laboratory (SoberIT)

Written by
Jarno Vähäniitty, Kristian Rautiainen, Ville Heikkilä & Kevin Vlaanderen
(see page v for details)

Edited by
Ville Heikkilä, Kristian Rautiainen & Jarno Vähäniitty

Cover art by Ville Heikkilä

Distribution:
Aalto University
School of Science and Technology
Software Business and Engineering Laboratory (SoberIT)
P.O. Box 19210
FI – 00076 Aalto
FINLAND
URL: http://www.soberit.hut.fi/sprg
Tel. +358 9 470 24851
E-mail: sprg@soberit.hut.fi

ISBN 978-952-60-3498-0

FOREWORD

by Sjaak Brinkkemper and Slinger Jansen

The software industry is going through radical changes. From new development technologies to changing delivery paradigms, the field has shown tremendous improvements to aid developers in achieving their goals and dynamically facilitating ever increasing requirements from a demanding market. Agile methods, such as Scrum, XP, and DSDM, have been introduced with the developer at the focal point, leaving behind those who manage the endless supply of requirements from the market.

The managers in charge of products are still holding on to ancient product management techniques, supplied by experts in the field of physical product management. These product management techniques do not sufficiently support software products, due to the extensive differences between physical and software products. Some examples of these differences are the fact that software is malleable, variable, can be released in rapid successive versions, and can be duplicated at no costs. Furthermore, there exists only a small range of physical products with such specific applications and such a wide range of stakeholders in its development and use as software products.

Software product managers are in need for supportive, modern, dynamic, adjustable, and transparent management methods and tools that are compatible with modern agile software development practices. This book is one of the first that actually provides concrete product management tools based on sound scientific principles, with the specific focus of improving a software product manager's practices in the fields of agile requirements management and portfolio management. The principles are not only timely but also comply with modern agile principles and in some cases even existing development practices and tools, without becoming too technical or academic.

The book that lies in front of you will provide product managers with the insights they need today to grow in an ever changing environment. It will help software product managers with expert knowledge and support tools for requirements and backlog management, release planning, and portfolio management. All based on extensive academic research and industrial experience. Furthermore, academics can, by reading this book, gain a quick overview of the state of the art in software product management. We applaud the authors with this wonderful result and hope this book will become a desk reference for all modern software product managers.

— Prof. Dr. Sjaak Brinkkemper and Dr. Slinger Jansen, Utrecht University

Why Should You Care About Agile Product and Portfolio Management?

Success in today's software industry requires integrating long-term product and business planning with technology development, juggling the scarce development resources so that those activities that from a business perspective are the most important get attended to, as well as combining flexibility and control provided by modern, agile approaches to software development, such as Scrum. However, this is not easy, and to be compatible with agile software development, the enterprise level processes of product and portfolio management have to be understood in a new way.

While there are plenty of books on product management, new product development portfolio management, as well as on agile software development, few authors so far deal with how product and portfolio management should be organized or even understood together with agile software development methods. Thus, as far as these books ignore the other side of the equation, they are actually a part of the problem.

While doing agile "right" is in principle simple, it is also extremely difficult. The cultural implications of a lean/agile transformation are immense, and require a lot of unlearning to take place for both individuals as well as organizations. As understanding how product and portfolio management can be made agile-compatible is not common knowledge, or even that well described in the latest literature, it is not a surprise that "adopting agile" can be a long and winding road.

In this book we provide a synthesis of guidelines from those relatively few authors out there that deal with the reconciliation of long-term product and business planning, portfolio management and agile software development. Combining these with our own findings from a decade of research collaboration with the top Finnish Software Companies, we hope you find this book a part of the solution.

How to Read This Book

This book is divided into three parts. For a quick start, read this page, see the table of contents (page x) and then, the introductions of each part (pages 1, 52 and 114). After that, you can skip back and forth as you feel like, for much of the book has been written so that the different chapters can be read relatively independently. For this reason, you may encounter some repetition – as well as a heavy degree of cross-referencing between the chapters.

Part I provides an introduction to this book by recollecting our earlier work on time pacing, as well as explaining the difficulties in fitting product and portfolio management together with modern, agile/lean approaches to software development.

Part II presents the Portfolio Management Health Barometer – a method for assessing whether your company needs to improve on its quest towards enterprise agility – as well as the theoretical underpinnings of the method. We also describe in detail how to use the method and its accompanying open source survey tool to conduct a health barometer assessment for your company – or for another company, should you be in the consulting business.

Part III presents our framework for agile product and portfolio management and selected practices regarding areas of agile product and portfolio management that have proved challenging in practice. Part III also summarizes key requirements for backlog management tool support for linking daily work with product and portfolio management.

Throughout the book we have used two kinds of boxes to highlight helpful information for those doing a quick skimming through the pages:

The boxes with a light bulb symbol provide additional helpful tips on the subject matter of the text.

The boxes with a warning sign symbol inform you about common pitfalls and dangers related to the subject matter of the text.

WHO WROTE THIS BOOK

This book summarizes findings from two research projects: First, ATMAN (Approach and Tool support for development portfolio MANagement), a research project funded by Tekes and the participating companies and conducted by members of the Software Process Research Group (SPRG) of the Software Business and Engineering Institute (SoberIT) at the School of Science and Technology of the Aalto University, Finland. Second, the writing of Chapters 11, 12 and partly 13 has been funded by the Cloud Software research program conducted in collaboration with Finnish universities and software companies.

The contributions in this book are indicated in the table below chapter by chapter.

Chapter 1: Using Time Pacing to Manage Software Development	Kristian Rautiainen & Jarno Vähäniitty
Chapter 2: Agile Product and Portfolio Management – Crucial for Competitiveness	Jarno Vähäniitty
Chapter 3: The Gap in the Literature	Jarno Vähäniitty
Chapter 4: The Portfolio Management Health Barometer	Jarno Vähäniitty
Chapter 5: Performing a Portfolio Management Health Barometer Study	Ville Heikkilä & Kristian Rautiainen
Chapter 6: The Health Barometer Tool	Kristian Rautiainen
Chapter 7: Agile Product Management	Jarno Vähäniitty
Chapter 8: Portfolio Management and Agile Software Development	Jarno Vähäniitty
Chapter 9: Agile Development Portfolio Management	Jarno Vähäniitty & Ville Heikkilä
Chapter 10: The Agile Requirements Refinery	Kevin Vlaanderen, Slinger Jansen, Sjaak Brinkkemper & Erik Jaspers
Chapter 11: Scaling Up Agile Release Planning	Ville Heikkilä
Chapter 12: Kanban for Software Development	Kristian Rautiainen
Chapter 13: Requirements for a Backlog Management Support Tool for Agile Product and Portfolio Management	Jarno Vähäniitty & Ville Heikkilä

How To Cite This Book

This is an edited book with chapters written by different authors. When you cite this book, we encourage you to cite per chapter to give the authors the credit they deserve. The citations per chapter are as follows:

Rautiainen, K. & Vähäniitty, J. 2010, "Chapter 1: Using Time Pacing to Manage Software Development" in *Towards Agile Product and Portfolio Management*, eds. V. Heikkilä, K. Rautiainen & J. Vähäniitty, Espoo: Aalto University, pp. 2-30.

Vähäniitty, J. 2010, "Chapter 2: Agile Product and Portfolio Management – Crucial for Competitiveness" in *Towards Agile Product and Portfolio Management*, eds. V. Heikkilä, K. Rautiainen & J. Vähäniitty, Espoo: Aalto University, pp. 31-37.

Vähäniitty, J. 2010, "Chapter 3: The Gap in the Literature" in *Towards Agile Product and Portfolio Management*, eds. V. Heikkilä, K. Rautiainen & J. Vähäniitty, Espoo: Aalto University, pp. 38-51.

Vähäniitty, J. 2010, "Chapter 4: The Portfolio Management Health Barometer" in *Towards Agile Product and Portfolio Management*, eds. V. Heikkilä, K. Rautiainen & J. Vähäniitty, Espoo: Aalto University, pp. 53-71.

Heikkilä, V. & Rautiainen, K. 2010, "Chapter 5: Performing a Portfolio Management Health Barometer Study" in *Towards Agile Product and Portfolio Management*, eds. V. Heikkilä, K. Rautiainen & J. Vähäniitty, Espoo: Aalto University, pp. 72-85.

Rautiainen, K. 2010, "Chapter 6: The Health Barometer Tool" in *Towards Agile Product and Portfolio Management*, eds. V. Heikkilä, K. Rautiainen & J. Vähäniitty, Espoo: Aalto University, pp. 86-101.

Vähäniitty, J. 2010, "Chapter 7: Agile Product Management" in *Towards Agile Product and Portfolio Management*, eds. V. Heikkilä, K. Rautiainen & J. Vähäniitty, Espoo: Aalto University, pp. 115-125.

Vähäniitty, J. 2010, "Chapter 8: Portfolio Management and Agile Software Development" in *Towards Agile Product and Portfolio Management*, eds. V. Heikkilä, K. Rautiainen & J. Vähäniitty, Espoo: Aalto University, pp. 126-148.

Vähäniitty, J. & Heikkilä, V. 2010, "Chapter 9: Agile Development Portfolio Management" in *Towards Agile Product and Portfolio Management*, eds. V. Heikkilä, K. Rautiainen & J. Vähäniitty, Espoo: Aalto University, pp. 149-156.

Vlaanderen, K., Jansen, S., Brinkkemper, S. & Jaspers, E. 2010, "Chapter 10: The Agile Requirements Refinery" in *Towards Agile Product and Portfolio Management*, eds. V. Heikkilä, K. Rautiainen & J. Vähäniitty, Espoo: Aalto University, pp. 157-169.

Heikkilä, V. 2010, "Chapter 11: Scaling Up Agile Release Planning" in *Towards Agile Product and Portfolio Management*, eds. V. Heikkilä, K. Rautiainen & J. Vähäniitty, Espoo: Aalto University, pp. 170-183.

Rautiainen, K. 2010, "Chapter 12: Kanban for Software Development" in *Towards Agile Product and Portfolio Management*, eds. V. Heikkilä, K. Rautiainen & J. Vähäniitty, Espoo: Aalto University, pp. 184-192.

Vähäniitty, J. & Heikkilä, V. 2010, "Chapter 13: Requirements for a Backlog Management Support Tool for Agile Product and Portfolio Management" in *Towards Agile Product and Portfolio Management*, eds. V. Heikkilä, K. Rautiainen & J. Vähäniitty, Espoo: Aalto University, pp. 193-210.

ACKNOWLEDGEMENTS

The authors would like to thank Tekes – the Finnish Funding Agency for Technology and Innovation and the Cloud Software program for funding this work. The ATMAN research project was a part of Tekes' Verso (Vertical Software Solutions) program. Without the funding provided by Tekes this book – as well as a number of international research publications, theses, as well as Agilefant – would not have been possible. Especially we would like to thank Matti Sihto and Kari Ryynänen. Matti represented Tekes in ATMAN's steering group. In addition to occasionally helping us to get back to the ground from the clouds researchers sometimes must live in, he also understood that sometimes research must come before everything else. Kari Ryynänen was invaluable in preparing the ATMAN project proposal in the phases preceding the funding decision.

We would like to extend our gratitude to the companies that participated and funded the ATMAN research project: F-Secure for being the first and most likely the most important partner company, EGET/PAF for full partnership for the first two years and follow-up partnership for the third year, IPSS for full partnership for the second year and follow-up partnership for the first and third year, and eCraft, Napa, Mipro and Tekla for follow-up partnerships. Whatever wisdom these pages might hold, it would not have been possible without the strong industrial connection we enjoyed during ATMAN.

Especially we would like to thank the following persons from the ATMAN partner companies: Janne Järvinen, Pirkka Palomäki, Mikael Albrecht, Kati Laine, Pasi Takala, Mikko Parkkola, Gabor Gunyho, Towo Toivola, Markku Kutvonen, Mika Lehtinen and Erkki Lepre from F-Secure; Joachim von Schantz, Andreas Perjus and Mikko Rusama from EGET/PAF; Terho Norja from IPSS (as well as the entire IPSS staff for patiently putting up with early versions of Agilefant and the accompanying enthusiastic Agilefant consultants), Nicklas Andersson from eCraft; Tom Sundell and Klaus Ihlberg from Napa; Suvi Hyyryläinen and Merja Koponen from Mipro; and Jari Sundqvist and Ritva Keinonen from Tekla.

We are deeply grateful for the contributions of the people who have during the project been a part of the ATMAN research project staff. Over the two first ATMAN years, Ilkka Lehto contributed significantly to the project both in terms of findings as well as by steering the development of Agilefant. Thanks goes also to Pasi Pekkanen and Reko Jokelainen who were able to turn Agilefant from a lumbering behemoth into the lean and mean war animal it currently is; Antti Haapala – an excellent "catch" by Prof. Pekka Kess of the Industrial Engineering and Management department of Oulu University – came all the way to Espoo to help us develop Agilefant's Daily Work functionality.

Five teams (2Rox, Maranello, Spider, Testarossa and Fiorano) took Agilefant further and further during their completion of the software project course at TKK. For a complete list of contributors to Agilefant, see www.agilefant.org.

We also owe thanks to Casper Lassenius, the responsible leader of the ATMAN research project for understanding, supporting and mostly tolerating our sometimes unconventional ideas and ways of working.

We would like to thank Maria Paasivaara for being the manager of the Solakka project during the year 2010. We would also like thank all the industry people and researchers we have met in Cloud Software program events for the many thought-provoking discussions.

In terms of international research collaboration, we have been very lucky. Prof. Dr. Sjaak Brinkkemper's group at Utrecht University performs world-class research in the area of software products and we are grateful we could participate in his group's work. In addition to Prof. Brinkkemper, we would like to also thank Kevin Vlaanderen, Slinger Jansen, Inge van de Weerd and Sandra Verdonk. The support, community and facilities they provided us during 2010 were invaluable for conceiving this book. Our trips to Utrecht have not only been a great learning and working experience, but at the same time, fun of epic proportions.

We are grateful for Prof. Dr. Günther Ruhe from the University of Calgary for his participation in ATMAN project. Ville Heikkilä is especially thankful for Prof. Ruhe for providing an excellent environment for the three months he spent in Calgary and for supporting Ville in writing of his first international research publication.

As always, our co-workers in the Software Process Research Group (SPRG) have been a great help in our research and they have provided invaluable commentary and guidance in their specialized areas as well as otherwise. SoberIT's support team has enabled us to work on our research and this book; we thank Johanna Lehtola, Miihkali Euro, Ritva Parvela and Jyrki Airola for their efforts.

Finally, last but definitely not least; our families and friends– you know who you are – thanks for allowing us go out to the frontiers of science, perform great deeds, and reminding us that also the most important activities need our attention every now and then.

CONTENTS

Part I: Introduction to Time Pacing and Agile Product and Portfolio Management

Part I of this book serves as an introduction to the topics of the book. Chapter 1 recollects our early work on time pacing that led our research to the topics of agile product and portfolio management. These topics are shortly explained in Chapter 2 and Chapter 3. Chapter 2 describes the problem setting that warrants the study and improvement of agile product and portfolio management. Chapter 3 discusses the gap in current literature between agile software development and the broader topics of software product management and portfolio management.

Chapter 1: Using Time Pacing to Manage Software Development

Kristian Rautiainen & Jarno Vähäniitty

We start this book by recollecting the background of our research, which has led us to the topics of agile product and portfolio management. This chapter explains the concept of time pacing and how it can be used in managing software development. Time pacing is in the heart of most agile software development processes and helps structure the work and collect fast feedback of the work and the working practices. The idea is to be both flexible and controlled. Flexibility is gained by doing things in short iterations, allowing for the possibility to react to changing circumstances in the marketplace or in the organization. Control is gained by not changing everything all the time, only at the beginning or end of a timebox, in a controlled way. Finding the right level of flexibility and control is a balancing game, where many stakeholders need to participate.

In this chapter we first provide motivation to why this topic is important (Section 1.1), then define time pacing (Section 1.2) and give an overview of time pacing on different so-called planning time horizons (Section 1.3). Implementing time pacing in your organization can be challenging, and therefore we provide a short introduction to organizational change management and software process improvement and give some time pacing implementation tips in Section 1.4. The chapter is rounded up with experiences from case companies that have implemented time pacing (Section 1.5).

1.1 Motivation

Managing software product development is challenging but doing it well can be extremely rewarding. Profits from duplicating a product to thousands or millions of customers can be both luring and elusive. Success in the product business demands more than just succeeding in individual development projects. Shipping products at the right time, hitting market windows of opportunity with

the right set of features over and over again is at least as important. However, in the software product industry, time-to-market is constantly shrinking and technologies evolve at a furious pace. If a company tries to keep up with this pace and react to every change in its environment, it does not have time to do anything else. The developers quickly go crazy with the indecision of the managers and the constantly changing product requirements. The key lies in striking the right balance between flexibility and control that serves both business and development needs.

Achieving this balance, however, is no easy task. For small companies (with less than 50 employees) which constitute the majority of software product businesses, it is particularly challenging. Many of these try to succeed in the product business, while at the same time doing customer projects to maintain cash flow. This leads to internal chaos, with people trying to do too many things at once. Projects exceed their budgets and schedules and only heroic efforts from individuals keep the projects going. Understanding the software process and using good practices might help, but everyone is too busy to stop and figure out what and how things could be done better. It is like a running man carrying his bicycle who is too busy to stop to mount the bike and pedal away.

The man carrying the bike has it easy compared to most small software product companies. At least, he has only two simple choices of action. For software companies a myriad of process models, methods and practices exist that could help improve development performance. However, as Frederick Brooks Jr. (1995) puts it, there is no silver bullet, no magic methodology that can solve all your problems. Choosing and tailoring processes and practices is difficult, especially since most processes and practices have been developed for and tested in large companies. For small software product companies that operate in turbulent environments, so called agile processes might be a good starting point. They have been designed for small teams and projects facing a lot of uncertainty. They provide a set of values, principles and practices that enhance flexibility and help you embrace change. If you understand these values and principles and are able to adopt the practices, you gain flexibility and retain control despite being flexible. However, real life has shown that it is easy to be too flexible and thus lose the control, as the following fictional anecdote demonstrates.

> Two weeks ago Jack, a senior developer, handled the installation for customer company Snoot Ltd. He is now working on a must-have requirement for an upcoming product release at the end of the development iteration. As he is taking a short break to stretch his muscles after an intensive programming session, a phone rings on Jane's table. Jane's tasks include, among other things, customer support. Unfortunately, she is at the grocery store downstairs to buy doughnuts for the company-wide Wednesday afternoon coffee break. Taking a brief look at Jane's ringing phone, Jack notices that the caller is Tom from Snoot Ltd. Tom was responsible for last week's

installation on the customer's behalf. Naturally, Jack is curious about how the company is doing with the delivered product, and answers the call on Jane's behalf. Tom thinks he has reached the helpdesk, and he tells Jack some improvement suggestions to some of the features he has had in mind and reports two suspicious phenomena he considers bugs. Jack listens and scribbles down Tom's observations on a post-it note he found on Jane's table. After the phone call, Jack returns to his computer and spends the rest of the day and a good half of the next enthusiastically programming two of Tom's improvement suggestions that he considers relevant. He also tries to reproduce and fix the bugs Tom had told about. On Thursday afternoon, Jack succeeds in fixing the second bug Tom mentioned and sends him an update. He then resumes programming the 'must-have' feature for the upcoming release. On Friday afternoon, in the weekly development team meeting, product manager Jeremy reviews what everyone has done during the week. He eventually finds out about the call Jack had intercepted on Wednesday. Jeremy is partly glad that Snoot Ltd. had an experience of an instant reaction to their needs, but he is mostly frustrated because the 'must-have' feature got delayed by modifications that are of questionable significance to the majority of customers. Jeremy asks Jack to provide Jane the details of those improvement suggestions he had not yet realized, so she can put them into the feature and idea database. Unfortunately, Jack does not remember the suggestions anymore. While the post-it note with the specs is still somewhere, it is likely that nobody (including Jack!) is able to decrypt Jack's scribbling.

A few weeks later Jeff, the CEO of the company, gets a brilliant idea to improve a certain feature in the product when making a sales pitch at prospect Boot Ltd. Returning to the office in the afternoon, he immediately tells his idea to a junior developer Joe, and asks whether Joe thinks the idea would be possible to be realized. After the conversation, Joe stops testing the feature Jeremy instructed him to test, and starts working on a prototype to find out if Jeff's idea could work. Two days later, Joe succeeds in demonstrating the validity of the idea. He runs to show the demo to Jeff, who is in the middle of a meeting with Jeremy about the status of the upcoming release. After refreshing Jeff's memory and receiving his commendation, the poor junior developer also gets scolded by Jeremy for his actions. Although Jeff's idea has now be demonstrated to work, one important feature remains untested. Furthermore, a more experienced developer could have demonstrated the feasibility of Jeff's idea in a couple of hours.

While the company in the anecdote displayed great flexibility, control was missing. The people might not have been aware of their roles and responsibilities, and they thought they were doing the company a favor with very fast reaction to customer needs. They also did not realize that they jeopardized the resource allocation of the development project thus risking future product releases and cash flow. The CEO and the management team had probably not created an explicit product strategy or roadmap for all to follow and thus there was no base-

line or vision to consider trade-offs against. While flexibility is good, too much flexibility can lead to chaos and therefore a certain degree of control is needed. Control should not stifle the flexibility and creativity needed in a software company operating in a turbulent environment. Instead, it should set the necessary constraints to prevent total chaos.

Time pacing is a key to flexible and controlled development and lies at the heart of agile software process models. Even before agile software development was introduced, time pacing had been proposed as a way to combine flexibility and control in reaction to changing circumstances. Time pacing means creating a pace for software development by dividing time into segments of temporal milestones at which part the functionality of the final software product is made ready[1]. In other words, the schedule is fixed and the scope is varied depending on the progress of the development team. The progress of the development work can be reviewed by all relevant stakeholders at these temporal milestones and decisions about future plans for the product can be made based on visible progress. This provides flexibility in changing the plans regarding the final product as we know more about the product and the market needs for the product, but also control since we cannot change plans all the time, only at the temporal milestones.

Brown and Eisenhardt's (1997) findings bring forth three key properties of successful organizations that need to change continuously in face of uncertainties in their business: (1) *semi structures* with clear responsibilities and priorities coupled with extensive communication, (2) *links in time* that direct attention simultaneously to different time horizons, and (3) sequenced, well-organized *steps of transition* from present to future projects.

Semi structures refer to organizations in which some details are prescribed and others are not. Among the prescribed details can be, e.g., project priorities, different roles and responsibilities, and agreed time intervals between product releases and iterations. Some structure is needed to facilitate coordinating change, but too much structure stifles the organization making it hard to react when needed. *Links in time* refer to organizational practices that handle past, present and future time horizons and the transitions between them. This could be, e.g., creating roadmaps to help envision a long-term product strategy and using these roadmaps to make prioritization and trade-off decisions for projects and iterations when planning them at the same time using history data to help estimate future workloads. Smooth transitioning between projects (and iterations) is vital to keeping the pace going. Time pacing already helps in providing regular and predictable time intervals of transition, but well planned and executed *steps of transition* are also needed. This could mean, e.g., a prescribed procedure for

[1] The term *ready* here implies that the system is in a stable and tested state.

iteration and project planning, so that the transition between iterations and projects goes as smoothly as possible.

Next we take a closer look at how we see time pacing. We have taken a lot of inspiration from the principles of the agile alliance[2] and different agile process models, especially Scrum and XP (Beck 2000, Schwaber & Beedle 2002).

1.2 Definition of time pacing

Time pacing means dividing time to be expended to achieving a goal into segments of temporal milestones at which progress is evaluated and possible adjustments are made to the plans. Changes are only made at these milestones, so persistence is accomplished at the same time establishing the flexibility to change plans and adapt to changes in the environment at the specific time intervals (Gersick 1994). These time intervals, or *time horizons* from a planning perspective, create a *pace* for product development. Time pacing also refers to creating new products or services, entering new markets, or launching new businesses according to a fixed schedule (Eisenhardt & Brown 1998). In contrast, *event pacing* refers to following a plan until something forces to deviate from that plan, e.g., moves by the competition or weakened performance.

Time pacing is not to be confused with the practice of scheduling regular meetings or milestones. The key to time pacing is in the adaptive behavior, which, if lacking reduces the meetings to mechanical cyclical events that can blind the managers from the need of change. Gersick (1994) uses the term "temporal maintenance" to describe these kinds of cyclical activities, as she sees their function as preserving the status quo instead of changing it.

Time boxing (Martin 1991, McConnell 1996, Highsmith 2000) has been used as a term in software engineering referring to doing time pacing on a project and iteration time horizon. Time boxing means that the end date of each iteration is fixed and the end date of the project is also fixed. Fixing the end date means that if you cannot finish all requested features by that date, the scope is reduced so that the system is ready at the fixed date. The requirements must be prioritized so that the team can make scope adjustments by themselves. This is important, because otherwise you could inappropriately force the team to work overtime in order to meet both the deadline and the scope. An important part of time boxing is freezing the requirements for the duration of an iteration. New features cannot be introduced in the middle of an iteration, except if an existing feature needs to be redefined because of, e.g., unexpected technical problems or misinterpreted user needs.

[2] http://www.agilealliance.org/the-alliance/the-agile-manifesto/the-twelve-principles-of-agile-software/

1.3 Overview of time pacing on different time horizons

Figure 1.1 shows an overview of different time horizons we have identified for time pacing. The most central part of time pacing is splitting development work into *iterations*. In each iteration functionality is added to the software so that we have a stable new version of the software at the end of the iteration. A number of iterations form a *release project* (3 iterations in the picture). The result of a release project is an internal or external release of the software. An internal release could be, e.g., an interim release to effort-intensive system testing. An external release could be, e.g., the final release to customers. The daily work within the iterations is coordinated and progress is monitored with *heartbeats*. These three levels of time pacing form the process for the development team. The longest time horizon is *long-term planning and portfolio management*, which spans two or more projects into the future. It deals with the long-term plans for the product and project portfolios of the company (the subject of the rest of this book), provides an interface between business management and product development, decides what release projects are launched and completed or killed, and prioritizes features for project and iteration planning.

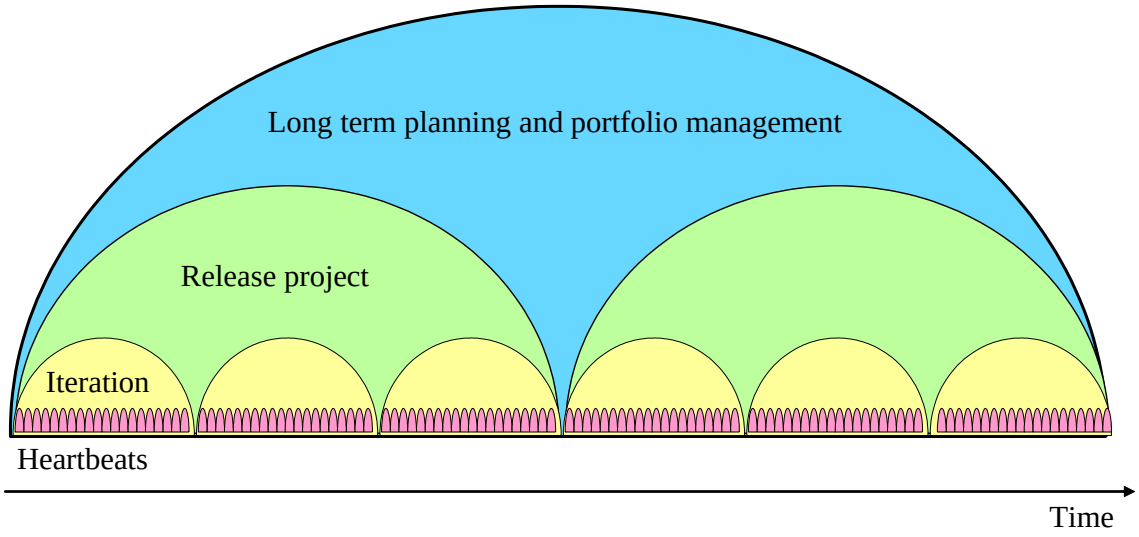


Figure 1.1: The different time horizons of time pacing

1.3.1 Iteration

The most central part of time pacing is splitting the development work into time boxed iterations during which incremental functionality is developed and integrated into the product. This forces you to divide the work into smaller pieces thus reducing the complexity of the work at hand. Also, it helps you to focus on what is essential to develop at that moment in time, forcing often quite hard trade-off decisions that might not normally get done.

When splitting the work and planning the tasks for the iteration you risk losing sight of ”the big picture” of what the end product should be. Keep the product vision in mind when planning and performing tasks. Let the vision guide your work and any possible trade-offs. Without a product vision you might get lost in all the details.

Table 1.1 shows an overview of a typical iteration. The iteration starts with iteration planning where the work to be done during the iteration is planned in cooperation between business representative(s) and the development team. Both are needed since iteration planning entails communicating the business and technical concerns to relevant stakeholders. The simplest possible outcome of iteration planning is an iteration backlog (used e.g. in the Scrum process model (Schwaber & Beedle 2002)) containing the vision, goals and tasks for the iteration.

Table 1.1: Overview of a generic iteration

Activity	Goal(s)	Participants	Result(s)
Iteration planning	Set goal(s) for the iteration Plan work to be done during the iteration (incl. testing!) Communicate business- and technical concerns to relevant stakeholders	Business & Development	Iteration Backlog
Iteration management	Monitor work progress Adjust scope, if necessary (Abort iteration)	Development (& Business)	Burndown graph Updated iteration backlog
Iteration review	Show iteration results (e.g. working software) to relevant stakeholders Get feedback for further development of the product	Business & Development	Working software Ideas for improving the product
Reflection	Reflect and improve on the ways of working	Development	Improved practices and processes

If all planning is left to the iteration planning meeting, the meeting will be very inefficient. A team's product owner should keep track of and continuously update a prioritized product backlog (containing features and ideas for the product with rough work effort estimates). The lead developer(s) could help the product owner by providing updated work effort estimates based on the latest development progress and so on.

Here is an example of a fairly rigorous iteration planning procedure that takes about a day to complete:

1. Defining the iteration backlog is done in an iteration board meeting. The iteration board consists of stakeholder representatives for different viewpoints of the product, e.g. the development team leader, the product owner, the chief of customer services, and a sales and marketing person. Inputs are customer and other commitments, the product roadmap/backlog and unfinished tasks from the previous iteration(s). The output is a *list of* desirable *issues* to be tackled in the following iteration.
2. Designing tasks for the iteration backlog from the *issue list* is done as group work by the development team, where the developers define the tasks that need to be done to complete the issues in the issue list. If any issue is unclear, it is immediately discussed with the iteration board members for clarification.
3. Estimating work effort for the tasks and checking availability of resources is done as group work by the development team, based on earlier experiences and existing and known commitments.
4. Prioritizing issues to complete or postpone is done by the iteration board based on the given work estimates and the available resources. The list of issues is typically longer than can be accomplished in an iteration. Choices must be made of what to include in the iteration backlog and what to leave to future iterations.
5. Committing to the iteration backlog is done by the development team. It reviews the choices made by the iteration board and decides on accepting the tasks or continuing the discussion if something seems unacceptable. If the development team accepts the iteration backlog it should next form a high level *iteration goal* for the iteration based on the iteration backlog.

In a turbulent environment the length of an iteration should not exceed one month, during which a stable new increment is developed and integrated into the product. During an iteration the requirements and resources should be frozen. Therefore, if it is possible to split work into shorter iterations without excessive overhead, it is advisable to do so to guarantee the availability of the allocated resources. The shorter the iteration, the fewer the possible interruptions are. The developers should be allowed to concentrate on the work planned for

the iteration. This should increase the efficiency and enjoyability of work. The key lies in including all known commitments that need attention from the developers into the resource allocation plan in the beginning of the iteration. For example, if Jack needs to help in integrating the system at a customer's site, the time needed for this should be subtracted from Jack's product development time, and so on. Even for a month-long iteration there should not be any big surprises, except for bugs found by the customers, and for this you might consider dedicating one person or allocating buffer time.

Do not integrate your increment into the existing product only at the end of an iteration in a so-called big-bang integration. Instead, try to integrate as often as you can. For more information on positive effects of integrating often and on implementing continuous integration, see e.g. Agile Software Development - Best Practices for Large Software Development Projects by Stober and Hansmann (2010).

Work progress is monitored at least in pace with the heartbeats during an iteration. This entails at a minimum updating estimated work effort left, possibly using a burndown graph. If the burndown graph shows that there is more effort left than there is time allocated for development, some corrective actions, such as reducing the scope of the iteration must be done[3]. For this to work, the prioritized list of tasks and goals in the iteration backlog must be constructed in a way that allows scope to be reduced by the development team. For example, the tasks for the accomplishment of goals should include 'must have' and 'nice to have' items, so that the scoping can remove some or all of the 'nice to have' tasks and still fulfill the iteration goal. If the iteration goal, however, is compromised, a meeting should be arranged between the development team and business stakeholders to decide how the iteration is to be scoped down. In case a showstopper problem (e.g. a major bug) appears in an already released product and requires lots of resources to handle, you should also consider aborting the iteration and starting a new one when the new resource allocation is clear. In any case the iteration management decisions and actions should be made clearly visible by updating the iteration backlog accordingly.

The iteration ends in a review meeting. The idea is to gather all relevant stakeholders and show what was accomplished during the iteration. A typical iteration review could contain a comparison of the plans to what was done and demonstration of working software. In this way development becomes visible to the stakeholders and comments and feedback can be gathered to help improve the product further in the upcoming iterations. The product can also be handed over to more time consuming system testing at the end of an iteration. Note that

[3] There are typically two other options. You could try to add more developers, but adding resources to already late project rarely works (Brooks 1995). Or you could extend the length of the iteration.

this does not imply that all system testing should be performed after the iteration!

The end of an iteration provides a good point in time to reflect on the ways of working for the development team. It is recommendable to have a reflection meeting and discuss what practices worked well during the iteration and what could have been done better. At this point it is also possible to introduce new or improved practices to try out for the next few iterations.

1.3.2 Release project

Time pacing on a project level is not necessarily always meaningful, but when strict deadlines are involved, e.g. hitting certain market windows, it is advisable. However, if you do not time box your projects, you have to place more emphasis on finishing the product at the end of each iteration. Besides helping in hitting market windows, time boxed projects force you to express and communicate your strategic intent regarding the product under work. Your resource allocation plans are an indication of your priorities, which should reflect your strategic intent. Project planning shows your resource needs and when many simultaneous projects are involved you are forced to make trade-offs in resource allocation between the projects, because there simply never are enough resources for everything.

Trying to get as many things as possible done at the same time is never a good idea. With many things going on at the same time your personnel is forced to almost continuously make context switches, which considerably slows down work. When planning projects and managing your project portfolio, try to advance a minimum set of important projects per iteration, so that you can get the most out of your scarce experts. In this way you actually get more done during a given time period than by trying to advance everything at once.

Table 1.2 shows an overview of a typical release project. Project plans in time paced projects are on a more coarse level than in more traditional, plan-driven projects. Detailed planning is left to be done during iteration planning. However, it is important to agree at least on project or release goals, including e.g. quality criteria. These should be based on a roadmap or a long-term release plan and then help guide the decision making and trade-offs when the actual work is done during the iterations and also when making the more detailed plans for each iteration. Depending on whether the release is internal or external, the quality criteria and release goals should differ, and they should be expressed explicitly. An internal release means that the release is only used within the company for e.g. resource- and time-demanding testing. An external release is what the customers get, for example, it could be a beta release for selected partners or a full-scale commercial release.

Table 1.2 Overview of a generic release project

Activity	Goal(s)	Participants	Result(s)
Project planning	Set project/release goals and vision Plan work to be done during the project (incl. testing) Initially allocate work into iterations Allocate resources Communicate business- and technical concerns to relevant stakeholders	Business & Development, Key Customers/Partners	Project/Release Backlog Initial resource allocation plan
Project management	Monitor work progress Adjust scope, if necessary (Abort project, if necessary)	Development (& Business)	Burndown graph Updated project/release backlog
Project review	Show project results (e.g. working software) to relevant stakeholders Get feedback for further development of the product	Business & Development	Working software, released product Ideas for improving the product
Reflection	Reflect and improve on the ways of working	Development (& Business)	Improved practices and processes

When planning a project (or an iteration), you might want to try planning from a testing viewpoint. First consider what you need to do to get something ready for (integration or system) testing as early as possible. Then consider what that means in form of tasks to be completed and possible regarding any interdependencies between the tasks. This way you might be able to find new ways and ideas for planning and working.

The time horizon for a release project could be 3-4 months, although the frequency of commercial releases might be wise keep around 1-2 times per year. If you have not decided on a generic iteration pace for your company then you should decide the pace of the iterations for the project. When you do this, you should at least consider how often you need concrete and visible feedback (working software), and how long it takes to develop a meaningful and valuable increment to the product. The point is to strive for uninterrupted work during each iteration by freezing the requirements and resources for the duration of the iteration. However, for portfolio management purposes it is recommendable to define synchronization points for the iterations of different projects so that everything does not start and end at completely different times. Otherwise changing resource allocations in the middle of projects may be very difficult. More on this subject follows in Section 1.3.4 and in the rest of this book.

In project planning you might consider setting themes for the iterations which help you envision the initial goals for the iterations. Also, you need explicitly allocate time and resources for testing the product, so that you can make sure that the quality of the released version is sufficient to make releasing the product possible. This includes allocating necessary rework time in the end of the project to fix any defects that rise in alpha and beta testing. The theme for the last iteration could be “stabilization of the product”, and no new features should be added during that iteration. Figure 1.2 shows an example of possible iteration themes.

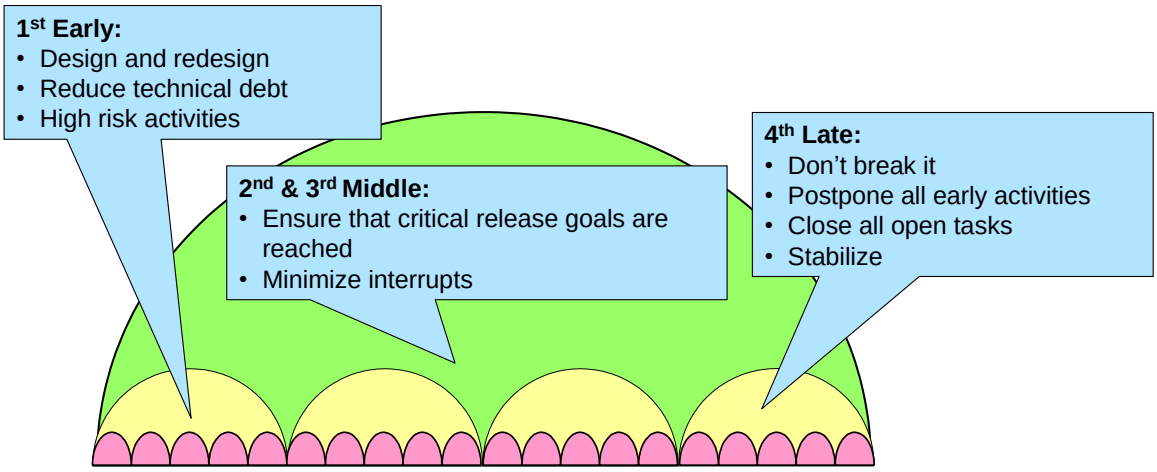


Figure 1.2 An example of iteration themes

Time-paced projects are managed by splitting them into iterations and managing the iterations. After each iteration you have a visible result that shows your progress. You can compare this to your expected progress and then make project-level decisions about corrective actions, for example, reducing the scope. If the project goal is endangered, then you might look at the portfolio level for resources that can be reallocated from other projects, or decide to dramatically reduce the scope and redefine the goals of the project.

Adding resources to an already late project might only make it more late, and at the same time compromise other projects! You should therefore be very careful when reallocating resources. (Brooks 1995)

As with iterations, a project ends with a review of project results and reflection on what went well, what did not go so well, and what could be improved.

1.3.3 Heartbeat

Iterations are paced with heartbeats in order to gain more insight of development progress. A heartbeat (or daily scrum meeting) is typically a status check that creates links in time from past to present to future in the form of three questions: What have you done? What problems are you facing? What are you going to do next? Such status check should not take more than 10-15 minutes per day.

Be careful not to start discussing revealed problems in the heartbeat meeting. Identify the problems, identify who can help solve the problems and move on, so as not to take up unnecessary time from the whole development team.

The time horizon of a heartbeat should be one day, but sometimes longer time horizons can be motivated. During a heartbeat the actual development is performed and at the end of the heartbeat the status is checked. In this way there is up-to-date information at regular, short intervals about project/iteration progress. This helps in identifying early warning signs of things that might compromise development goals. For example, if Joe has not been able to use his time as planned to developing the product, this is revealed in time for corrective actions to be taken, instead of being revealed at the end of the iteration when it is too late to react. Or if Joe reports that he is still working on the same task as three days ago when the original effort estimate was 5 hours, somebody else will notice a potential problem and it can be addressed. If the effort left estimates of tasks are not updated continuously, they should be updated at least once a heartbeat cycle. A part of synchronizing the work might be making daily builds and running automated smoke tests against them. This gives an indication of

system status from a technical perspective. Also, if you have separate testers or a separate testing team, heartbeat meetings are a good place to follow up the development progress, helping testers to plan and synchronize their own work with the work of the developers.

1.3.4 Long-term planning and portfolio management

Long-term planning and portfolio management sets the direction for product development by aligning the product development efforts with the business and technology strategy of the company. This means considering the overall strategic ambitions of the company together with the competences and availability of people that do the actual work in conjunction with planning future releases of products. You may wonder if there is any point in making long-term plans in a very turbulent environment, since the plans keep changing anyway. These coarse plans force you to explicate your current understanding of where the company and its product are moving and thus provide you a point of comparison for trade-off decisions which might need to be made. For example, when you face a decision for allocating your resources either to "quick cash" from a newly emerged customer project X or continuing to develop the next "killer app" according to your long-term plans, you actually can make an informed decision as long as the long-term plans are available. Even if you desperately need additional cash you may be able to steer customer project X into a direction that also supports your long-term plans.

For the upcoming 2-6 product releases you should plan the release projects on a high level of abstraction (e.g., product vision, major new features and technology, quality goals, release schedule, coarse resource allocation) and document them, for example, in the form of an aggregated release plan or a product and technology roadmap. In this way you can create a baseline against which to you can make trade-off decisions. For example, if a customer makes a request for something that is already in the roadmap, you can ask if the customer can wait until the planned release in 4 months, instead of immediately altering existing plans and disrupting the work already in progress. If the customer still insists on speedy delivery of the requested feature, you may at least be able to negotiate a premium to the price.

Christensen and Raynor (2003) pointed out the crucial role of the resource allocation process in putting a company's strategic intention into action: *"...a company's strategy is what comes out of the resource allocation process, not what goes into it."* This means that besides being time paced with, for example, major roadmap revisions every 6-12 months, long-term planning and portfolio management should be represented in resource allocation decisions at least on the time horizon of an iteration, since the outcome of iterations is what the company actually does. Since there are other activities possibly demanding product development resources, such as maintenance work and assisting in customer

deliveries and integration, resource allocation decisions can be difficult trade-offs. In a way, portfolio management can be understood as the work of constantly monitoring what happens inside and outside of the company so that necessary adjustments to the release plans and resource allocation can be made.

1.4 Implementing Time Pacing

Implementing time pacing in your organization means changing your processes and the way you work, which is always challenging. You need to know the basic principles of software process improvement (SPI) to successfully execute such an initiative. SPI is also about organizational change. In this chapter we shortly present some important aspects of organizational change and the basic principles of SPI. After that we give hints on how you could approach your own implementation of time pacing.

1.4.1 Organizational change

Organizations are complex systems and therefore it is logical that organizational change is complex too. At least four organizational elements can be found in each organization according to organizational change research. These are shown in Table 1.3.

Table 1.3: Elements of an organization

Task	Structure	People	Technology
Practices	Decision making	Skills	Tools
Procedures	Role responsibilities	Knowledge	Techniques
	Coordination	Needs	Infrastructure
	Communication	Motivation	
	Management		
	Work flow		

The *task* is ultimately producing the products and services, including practices and procedures needed to perform the task. *Structure* is how the work is coordinated, how decisions are made and by whom, communication systems, or how the work is managed. *People* (or *actors*) refers to the ones performing the tasks and *technology* refers to the tools and techniques used, including the needed technological infrastructure for everything. The list of things under each element in Table 1.3 is by no means exhaustive, but rather serves as an example.

The four organizational elements are highly interdependent, which means that a change in one results in compensatory or retaliatory changes in the others.

Compensatory change means, for example, that the other elements change into something new, probably unexpected, to compensate for the change in one element. Retaliatory change means that the other elements try to prevent the change and force a status quo or even a negative effect to the changed situation. Therefore one should address all four elements in order to achieve permanent change. These four organizational elements are useful when you are planning a change initiative, such as software process improvement. They are also useful when performing a retrospective analysis of why things did not go as planned.

For example, when you are implementing time pacing you are primarily addressing the structure element. If you simply say ”hey let’s start doing time pacing” without defining the needed structural elements such as role responsibilities, decision making hierarchies, and communication patterns, you probably will not get very far. However, even after perfectly addressing the structural element you may fail because your existing practices and procedures may not work well in a time-paced process, or because your personnel might not possess the right skills or motivation for time pacing or the needed new practices and procedures, or your existing technology might not support any of the above. On the other hand, if you try to get all the elements right the first time around, you might never get out of the planning phase of your change initiative.

1.4.2 Basic principles of software process improvement

Commitment must start from the top

A key success factor for a SPI initiative is management commitment. If the top management is not 110 % committed to the SPI initiative, their actions easily hinder all efforts by others. By 110 % commitment we mean that just deciding that an SPI initiative is something that the organization should undertake is not enough. Management must also participate in every possible way in the initiative itself, for example, by planning their own management processes and practices simultaneously with the development processes. We have experienced that otherwise the organization’s processes do not necessarily work, as the following example illustrates.

> At HardSoft Ltd the developers were fed up with the way things worked. Almost every day Joanna (Sales Director for Gadget) called Jill (the Development Team Leader) and told her about the latest changes to the release plans, because she had sold a new feature to a customer. The developers never got anything ready by the release deadline, and planning was becoming more or less impossible. Jill had just had lunch with Jeeves (a Process Consultant) that she knew from when she studied at the University, and he had told her about time pacing. Jill got really excited and she convinced Jeff (the CEO of HardSoft) that HardSoft needed to improve their processes by implementing time pacing. A meeting was arranged between Jeff, Jill, and Jeeves to discuss it further, and after the meeting Jeff agreed that this was

definitely the thing to do. “I support you 100 %, go ahead with the SPI initiative”, said Jeff and the meeting ended.

Jill was very excited and so were the developers when she told them. During the next month they all participated in specifying the processes for HardSoft’s product development. Some design and implementation practices were adopted from eXtreme Programming for the heartbeat time horizon activities, the iteration length was specified to be one month, and a strict requirements and resource freeze for the iteration time horizon was decided and agreed with upper management. The roles and responsibilities of different stakeholders were also specified for each time horizon. A couple of training sessions were arranged to communicate and train the new way of working to all, including upper management.

The following month the new process was piloted in one of the release projects. To Jill’s disappointment, Joanna still kept calling her almost daily, insisting on changing the iteration plans, even though she had agreed to the new process with requirements freeze for the iterations and she had participated in the iteration planning meetings. Jill complained to Jeff, who was very sympathetic and promised her his support. However, during the following 6 months the situation did not get much better, not until John (the Visionary and vice CEO) was appointed as the head of the newly formed Product Management Board (PMB). The PMB was responsible for all the products of the company, including that the releases were successful and reflected the strategic ambitions of HardSoft. John used a couple of weeks to define the management processes for the PMB, and ended up with exactly the same processes as defined by Jill, the developers, and Jeeves, with the exception of some refinements and choices of words that he used from a management perspective. When these processes were applied in the following iteration, things started to work out, improving in each subsequent iteration, as the organization learned and adapted its processes. Jill and the developers were finally satisfied.

Improvement requires practitioner buy-in

Although management commitment is crucial, because of the resourcing issues involved, practitioner buy-in is at least as important. If the developers are not motivated to change their current way of working, they resist change causing inertia to the SPI initiative. Buy-in of the most respected and influential people are a definite must, since the others can be expected to look up to them and follow their example. One good way to ensure buy-in is to involve the developers in designing their own processes.

SPI must have a champion

Unless someone steps up as the leader and champion of SPI, other tasks easily take precedence over SPI. The champion needs to be in a powerful position to be able to hold his own in arguments with senior management and needs to have the respect of the developers or they will ignore him/her. That is why you should

not appoint a junior developer or manager as the SPI officer. The champion is often the most vigilant person to reinforce the process, which can easily be observed as decay in process conformance when he/she is absent. Different new practices, tools, or technologies that are introduced might also each have their own champion. The champion is the person from your organization who introduces the practice and trains and inspires the others in using the practice. These champions are nearly as important as the overall process improvement champion.

Improvement requires investment

Any improvement or change is going to cost you, both directly and indirectly. The direct costs stem from the effort used to, e.g., planning, training and documenting the improvement, or buying new tools. Indirect costs stem from the performance downfall that is typical to any improvement or change. When the people learn new ways of working, their performance is at first impeded, but when they learn the new (and hopefully) better way of working, the rewards in performance increases should cover the costs. There is no guarantee, however, that this will happen, so there are always risks involved in improvement. But in our opinion, the risks of not improving far outweigh the risks of improving in most cases. One way to minimize the risks is to pilot the improvements in a smaller scale, for example, only one development team could try out a new practice like pair programming. If the piloting proves successful, the new practice can be rolled out to the whole development organization.

First understand the current process

A common mistake in SPI is to specify an ideal process for the organization without considering the existing process. This might discourage the practitioners from even trying to reach it since it seems so different from the current process. The key is to start by characterizing the current process and identifying needs for improvement. One good framework for SPI, which includes this idea, is the Quality Improvement Paradigm (QIP), originally developed by Basili, Caldiera, and Rombach (1994) and shown in Figure 1.3.

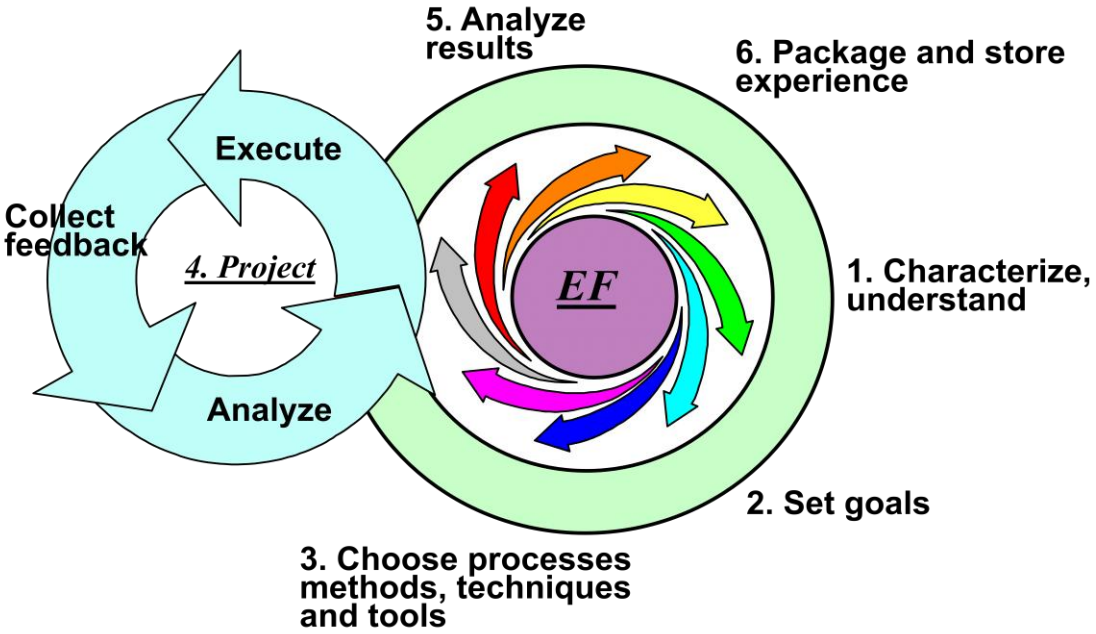


Figure 1.3: The QIP framework

The steps of the QIP seen in Figure 1.3 can be translated for SPI as follows:

1. *Characterize and understand* the current process(es) based upon available models, data, intuition, etc. Create a baseline for improvement using this characterization.
2. *Set goals* for the improvement initiative based on the characterization and understanding you have formed in Step 1. Remember to consider what has strategic relevance to your organization. The goals should be quantified (i.e. can be measured), so that you can assess the success of the SPI initiative based on how well the goals are reached. Do not set the goals too high or they might discourage your personnel. The idea is to improve the processes incrementally, so you should explicate short-term goals. A good idea is to include a long-term vision to help you in setting the short-term goals.
3. *Choose processes, methods, techniques and tools* for improvement on the basis of the characterization and goals you set. It is advisable to choose smaller parts for improvement instead of trying to improve everything at once, but still try to consider all four organizational elements to increase the likelihood of success in the improvement initiative.
4. *Execute* the project with the chosen improved processes, *collect feedback*, and *analyze* the feedback after each iteration to make further improvements.
5. *Analyze the results* of the improvement efforts when the project is finished. Use the data to evaluate the success of current practices and to determine problems, and document the findings and make recommendations for further improvement.

6. *Package and store experiences* in the form of new or updated models, checklists, instructions or other forms of structured information. The QIP suggests using an Experience Factory (EF) repository for this, but you can use whatever way of storing the information you are comfortable with. The main point is that the information is available to all stakeholders and that they are aware of its existence. Therefore a short training session for the improved parts of the process might be in order after each release project.

After this the QIP cycle starts from the beginning with new areas of improvement.

Sometimes you need to set your improvement goals extremely high to force thinking ”outside of the box”. This may be the only way to make significant improvements. However, keep an eye on the feelings of people to ensure that the effect is positive instead of discouraging.

Do not underestimate the importance of feedback for motivation

One of the best motivators for SPI is giving and receiving feedback. Feedback is especially appreciated by the developers. You should, for example, reward developers for a job well done and for process conformance. As in using QIP, you should create measurable goals for improvement and collect data during the improvement to see if it works. The results should then be presented to the developers and the implications discussed, for example, in reflection meetings.

Change must become a way of life

Processes that are not constantly improved die or decay. That is why change and SPI must become a way of life. The QIP presented above is good also in this respect, since it promotes cyclical, continuous SPI. A good way to complement the QIP is to have regular reflection meetings, where different stakeholders gather to discuss what has been working and what has not. In this way new ideas for potential process improvement can be gathered and everyone is kept involved with SPI, which also helps in creating better buy-in and increases motivation.

Institutionalizing improvement requires periodic enforcement

The principle of periodically enforcing the improved process to make it part of the organizational culture – institutionalizing it – is very close to the principle of change becoming a way of life. The main difference is that institutionalization is needed to make the changes more permanent, and *too much change* can prevent this. The best way to enforce the processes is peer pressure. Everybody is responsible for pointing out lapses in process conformance, and if there is a need to change the process, it should be addressed at the reflection meetings. As with freezing the requirements for iterations, the process should be frozen for iterations as well.

1.4.3 Finding a suitable pace

Figure 1.1 shows the different time horizons for time pacing that set the pace for product development, and that can be used as a starting point for planning and mapping different practices and activities to that pace. If a practice or activity belongs to a certain time horizon it means that it is planned and tracked with that pace. If necessary, a practice or activity can be split into parts to be tracked on a faster pace. To give you an example, let us consider how requirements can be managed using *backlogs*, an idea presented in the Scrum (Schwaber & Beedle 2002) process model.

> A bunch of product stakeholder representatives participate in the April roadmapping session to plan future releases of the products of HardSoft. Jeff (the CEO) represents the company strategy and wants to secure that the products reflect this strategy. John (the Visionary) provides some "out there" visions about the future development of the markets and technology, supported on the technology front by Jenny (the Chief Architect), who also is responsible for more "down to earth" assessments on the viability of using new technologies. Jermaine and Joanna (the Sales Directors) represent the customer viewpoint and bring the latest information from the markets. Jeremy and Jay (the Product Managers) are responsible for one product each and also represent the viewpoint of customer support (Help Desk and Product Delivery). Jericho (the Marketing Director) wants to secure sexy features to future product releases, so he can market the product successfully.
>
> The meeting starts with Jeremy (Product Manager of Widget) presenting the up-to-date product backlog of Widget, the older of the two products of the company. A product backlog is a prioritized list of product requirements and features of varying scope. All the ideas for the product have been gathered into the product backlog and Jeremy is responsible for keeping it up-to-date. Jeremy presents his preliminary suggestion for the release backlogs of the two following releases of Widget. A release backlog contains the requirements and features to be included in a product release and is a prioritized list, like the product backlog, only a bit more detailed. At the end of August a minor release of Widget is scheduled, containing some bug fixes and new features. The next major release for Widget is scheduled just before Christmas and contains support for new databases that are needed to penetrate new markets and a bunch of other new features.
>
> Jeff (the CEO) is pleased with the release backlogs, especially since the strategic intent of the company is to move to new markets to generate new cash flow. Jenny (the Chief Architect) expresses concern for the tight schedule, because Jack (the Senior Developer) has been very busy with rewriting Gadget (the second product of the company) for .NET, and the progress has not been as good as expected, as everybody could see in the last iteration demo of Gadget. Since Jack is the database expert of the company, a decision must be made on which is more important, getting .NET Gadget out in

time or extending database support for Widget. Jermaine (Sales Director of Widget) and Joanna (Sales Director of Gadget) argue that both are very important for their customers and Jericho (the Marketing Director) shows that market research results support an aggressive strategy to move into the new markets immediately. Before the meeting breaks into on open fight, Jay (the Product Manager of Gadget) proposes that he shows the release plans for Gadget, so that the possible trade-offs are clear to everybody. When Jay has shown his suggestions for release backlogs for Gadget, the lively debate continues until lunch. Everything seems important, and no trade-offs can be made.

After lunch, when things have cooled down a little bit, Jenny suggests that Jack continues working on porting Gadget to .NET. But, instead of doing it alone, he pair programs with Jo (a Junior Developer). Pair programming has been used earlier with some positive results in different tasks, so Jack and Jo already have some experience in doing it. This way Jo would learn from Jack and in a couple of iterations Jo would be able to continue on her own, if necessary, leaving Jack free to start working on the new database support for Widget. The drawback is that some of the features in the release backlog need to be reprioritized to lower priority, since Jo cannot work on them, meaning that they probably cannot be finished in time for the release. But at least the most important goals for the releases of both products have a greater chance of being met. The meeting participants discuss Jenny's suggestion for a while and agree that this is the best course of action. The meeting then continues with more discussion and reprioritization of the release backlogs.

A week later Joanna, Jay and Jenny meet with Jill (the Development Team Leader), Jack, Jo, Joe (another Junior Developer), and Jake (the Quality Engineer) to plan the next iteration of Gadget. At the coffee table they have already discussed some of the ideas from the roadmapping session, so there are no big surprises for anyone. Joanna, Jay, Jenny and Jill have prepared for the planning session by discussing the most important release backlog items and what they mean in more detail, both from a business perspective and from a technical perspective. The results from these discussions are presented to the development team and questions about unclear things are asked and answered. Then the team is left alone to plan how these release backlog items can be broken down into tasks for the next iteration and the effort to complete the tasks is estimated. When the planning is ready, the development team presents the results to the others and also discuss the budget of available development effort for the iteration. It is apparent that not all tasks can be done within the available budget, so Joanna, Jay, Jenny and Jill discuss and prioritize the scope of the iteration. They also create the iteration backlog, which contains the iteration goal(s) and the features to be done including the planned breakdown to tasks for developing those features. When the iteration backlog is ready, the development team joins the others and the backlog is shown and discussed. After that the team accepts

responsibility for realizing the iteration backlog at least to the extent that the iteration goal is met.

A month later the same iteration planning procedure is repeated using what was left in the release backlog as a starting point. At this time, new, emerged requirements can be traded off with those in the release backlog to reflect changed priorities. These should not, however, be in contradiction with the release goals. Changing the release goals every month would probably be overreacting to changes in the market. Of course, if the initial analysis of the markets was totally wrong, even the release goals should be changed.

The example above includes many issues of software product development management. To recap the main issue in the example, requirements can be managed using backlogs of different scopes as depicted in Figure 1.4. As we move to shorter time spans in Figure 1.4 the backlogs get more detailed. Managing the high-level requirements on a monthly pace gives us flexibility to change plans if we have missed something earlier. Each month we also see how much has been accomplished, giving us a measure of progress we can compare to the plans and goals. This gives us control to make corrective actions based on real progress if our original plans were too optimistic or pessimistic.

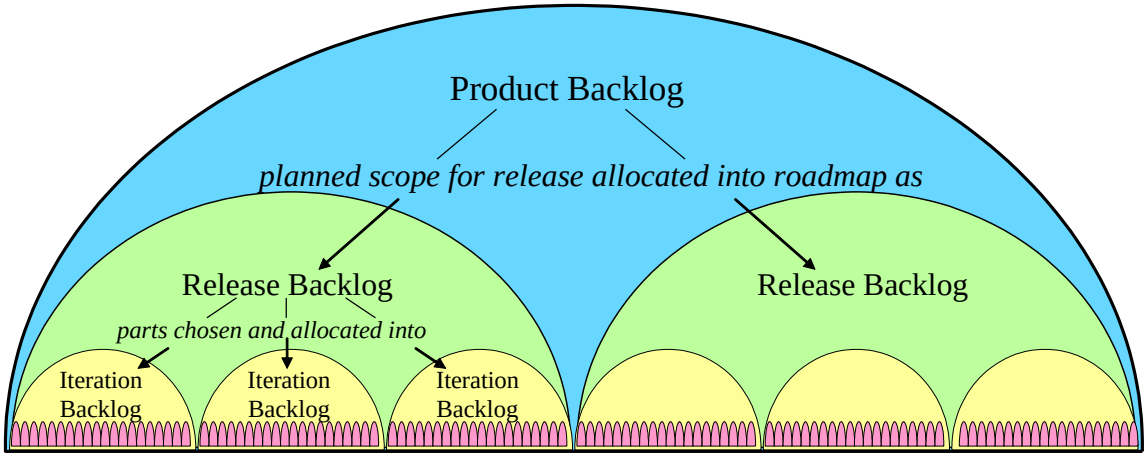


Figure 1.4 Managing requirements with backlogs

Finding a suitable pace entails understanding the rhythm of the markets and the internal capabilities of the company. Releasing products to the markets should be done with appropriate pace. For example, if a magazine publishes a product review at a certain time of the year, you need to release your product in time for that review. Or if a trade show is organized at a certain time, you need to have a product release ready by that time. Another example could be seasonality, for example, if you need to release a product for the Christmas market. Your product's maintenance agreement may also contain promises of maintenance releases with a defined pace. All releases of the product do not need to be external or commercial releases. You can also make internal releases that can be used in demos for potential customers or just used as intermediate versions for tho-

rough testing. This way you can get a better understanding of the product and improve it before you make it widely available.

While the rhythm of the markets tells you when you would want to release your product(s), the internal capabilities of the company constrain what is possible. With the internal capabilities we mean, for example, how effective your processes are, how skilled your employees are, how easy it is to develop and test your product(s) iteratively and incrementally, and how much development effort different people can contribute considering all the other tasks at hand. One way to find out the internal capabilities is to define and try some pace and see how well it works. For example, if you decide to make one commercial release of a product per year, you could make two additional internal releases per year. In this case the time span of each release is 4 months. Then you could define 1-month iterations for developing the product and use a daily heartbeat rhythm to monitor progress. If the tasks start taking almost an iteration to complete (instead of taking from one day to one week), you have not been able to plan and split the backlog items into small enough tasks. This could mean that you have selected too difficult and large features to be done in one iteration or that the iteration and possibly the heartbeat is too short. If you think the iteration and heartbeat pace is appropriate, you need to develop your skills in planning the iterations and the tasks to be done in them. One of the problems may also be that you do not yet understand what you are supposed to get done or how it can be done using some new technology[4]. In that case you might need to reconsider the release goals and iteration contents to reflect that you are learning a new technology. Pacing helps you show progress or lack thereof, which in turn helps you make informed decisions about continuing or discontinuing pursuing the goals or turning to an alternative course of action in order to be able to make the commercial release.

A more structured way of planning and defining the development pace based on the internal capabilities is considering what needs to be accomplished by the end of each time horizon and how long that will take. For example, when a product is released to the market, there is much more involved than just coding and testing the product. You may need product documentation, such as installation instructions and a user's manual. You may need marketing material containing, for example, screen shots of the product, well in advance before the product release. You may need sales material, such as brochures and demonstrations of the product, for the sales people. Such things have lead times that need to be considered when planning the iterations of the release. A good idea is to dedicate at least the last iteration before a commercial product release to stabilizing the product and finalizing all the necessary accessories. Stabilizing the

[4] Extreme Programming suggests using so-called technology or research spikes when you are not sure how something should be done (see e.g. Extreme Programming Explained: Embrace Change by Beck (2000)).

product means that we do not develop new features but rather make sure that the existing ones work properly. For this we need to do some testing and bug fixing, the amount of which depends on, for example, how much and what kind of testing we have been able to do in the previous iterations. If we need to do extensive testing that could take 2-4 weeks to complete, a 1-month iteration is not long enough to accomplish both testing, bug fixing, and re-testing. Also, for the screen shots for the marketing material, you may need to make a visual freeze earlier than in the last iteration.

You might also consider different pacing for different types of projects or activities. From a portfolio management and resource allocation viewpoint it is then important to try to synchronize the different paces. For example, if the iteration time horizon of a major product release project is 1 month (or 4 weeks), other iteration time horizons (e.g., for maintenance releases) should be 4 or 2 weeks, or even 1 week. In this way they are all synchronized at least every 4 weeks, which is then the pace for making major resource allocation decisions.

1.4.4 Adopting a practice

To give you an example on using the different time horizons in planning of a software engineering practice adoption, we will use refactoring. Refactoring means changing the internal structure of the code without changing the external behavior of the software. Refactoring is one of the key practices in Extreme Programming (XP) and you might be interested in trying it out. You could directly try the XP way as described by Beck (2000) or you might want to consider other approaches. Figure 1.5 shows three different approaches to refactoring that are done on different time horizons: 1) refactoring heartbeat, 2) refactoring iteration, and 3) refactoring release.

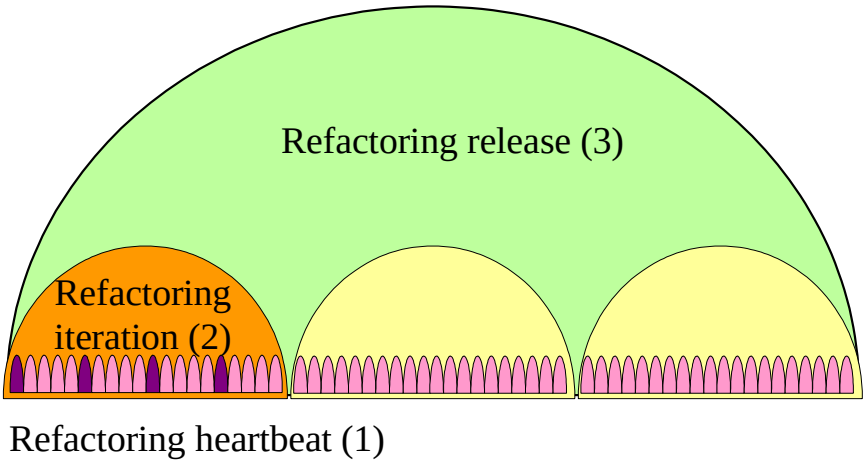


Figure 1.5 Different approaches to refactoring

1. Refactoring on a heartbeat time horizon could mean dedicating one day of the week to refactoring the code, as shown in Figure 1.5. It could also mean that refactoring is an integral part of the development work, as explained in

XP. Each developer is responsible for refactoring code when it seems appropriate.

2. Refactoring on an iteration time horizon could mean that the first iteration of a release project is dedicated to refactoring the existing codebase, which is shown in Figure 1.5. Another option would be dedicating the beginning of each iteration or several iterations to refactoring the code.
3. Refactoring on a release project time horizon means dedicating a whole release to refactoring the existing codebase. This option is probably the least likely to be used, because if the code quality is so bad that a whole release project is needed to fix it, you might as well rewrite the whole software.

The approaches above are by no means mutually exclusive. You could combine them, for example, by doing major refactoring in the first iteration of a release project and then in the rest of the iterations you do refactoring on a case-to-case basis and decide in heartbeat meetings if and when refactoring is needed. As described in this example, you have several options when you adopt a practice. You should go through same kind of reasoning when deciding how to apply a new practice.

1.5 Company experiences

We have worked with several companies helping them apply time pacing in their software development. In this section we summarize some of the experiences from these companies.

1.5.1 Resulting processes

All companies used elements from Scrum (Schwaber & Beedle 2002) in their software development process. Software engineering practices were picked from XP (Beck 2000) in some companies, while others picked practices from the Rational Unified Process (RUP) (Kruchten 2000). One company even included elements from the StageGate™ model (Cooper 1994) for their management process. The chosen time horizons varied a lot, as can be seen in Table 1.4.

Table 1.4 Time horizon length variation in companies

Long-term	Project	Iteration	Heartbeat
2-12 months	3-4 months	2-4 weeks	Daily-weekly

1.5.2 Elapsed calendar time to get the process up-and-running

It is fairly impossible to say exactly how long it takes for a process to be up-and-running, because you normally deploy a process incrementally. As in any change

initiative, the point where the process actually works as intended is very hard to pinpoint. Here we provide a few experiences.

In Company A, a small company where all 14 employees participated in planning and defining the process, it took one and a half months to start using the new process with partial features. As a new test manager was being hired, system testing details were left out from the first version of the process. Instead, the first version concentrated on requirements management and communication between the different stakeholders in the product development process. However, all practices from the first version were still in use 3 years later, which means that the first version can be considered to have been very successful.

In Company B iterative development (without time pacing) had been introduced by the new product development manager, who was the primary person to apply time pacing. It took only 1 month to start piloting time pacing in iterations, but it took more than 2 years for the process to stabilize. Even then there were still problems with long-term planning and portfolio management. The reasons for this are shortly discussed in Section 1.5.5.

In Company C it took one and a half months to get a defined iteration up-and-running, but it took almost a year to get portfolio management included in the process and even more time to get it really working. The project in which the iteration was piloted had started before the change initiative, which made some decisions in iteration planning harder, since an overall project plan to support iteration planning was missing. However, the definitions for iterations and projects remained in use after the pilot, which makes it somewhat successful.

All in all it can be said that it takes 1-2 months to plan and prepare for a pilot implementation of a time paced process, after which it can be incrementally improved. Sometimes it can take a long time to really get the process to work as intended.

1.5.3 Overall impressions of practitioners

One of the most important aspects of process improvement is to actually get people to use the new process. In all companies we have worked with the processes were used and liked by the personnel approximately 3 years after the process improvement started. The most common positive remark from the practitioners was that the process had practical utility and helped them in their work as well as helped them identify problem areas in the process. The time paced process actually encouraged people to do something about problem areas in the process, because they felt they could (and should) do things better when the process had become more visual and concrete.

Flexibility and control had been achieved in the companies. Many commented that their work was better planned and controlled than before. This was partly due to the demanding pace that forced people to plan their work more often and

partly due to adopted practices to planning that facilitated communication between developers and business people. Also, freezing resources and requirements for the duration of an iteration pleased the developers giving them a chance to concentrate on the tasks at hand.

1.5.4 Success factors

The companies that best succeeded in following the basic principles of SPI had the best experiences of success. On top of the list of success factors we can find management support, process/practice champions, and involvement of many developers in planning the process to ensure buy-in.

Some interesting catalysts of success also deserve to be mentioned here. In one company the process did not seem to work until one of the developers was involved in developing a process support tool especially intended to support the time pacing framework presented here[5]. This tool was then deployed in the company and it “forced” the people to use the process and made the process visible to all. The developer also emerged as a tool and process champion who could help and support the others, which made using the process much easier.

In another company, portfolio management had not been successfully included in the process until a new human resources manager was employed who was genuinely into managing people instead of participating in the technical work. It took him only a few weeks to kick portfolio management forward, at least from a resource allocation perspective, by introducing a resource allocation procedure where project managers had to ”buy” their resources every two weeks. Although this may sound harsh, it actually worked well for the company after an adjustment period of a couple of months. In this way the company could refocus their resources in a controlled way every two weeks, if necessary.

1.5.5 Factors impeding deployment

The most prominent factors impeding deployment of time pacing was lack of time for process improvement and too few participating people, which both show a lack of investment in process improvement. The common symptom of this was that key personnel had too many other things to do and process improvement suffered from low priority and did not advance. This affected the visibility of the process to all other personnel and the feeling of urgency in adopting the new process.

In Company B, introduced in Section 1.5.2, especially long-term planning and portfolio management was problematic. The company’s mission and business priorities were unclear and changed often, which was reflected as difficulties in focusing development effort. Since it was hard to agree on development priorities, there was no solid ground for portfolio management, which started to seem

[5] http://www.agilefant.org

futile. However, when the project and iteration-level processes started working there was an increasing demand from the developers for improved portfolio management, because without clearer development priorities they had to constantly re-plan their work, which felt as a waste of time.

In one company, management support was not whole-hearted, which manifested itself as disturbances and new requirement demands in the middle of iterations, even after everybody had agreed on the rules of freezing the requirements. The process started working as intended only after management designed its own processes for iteration planning and portfolio management.

Chapter 2: Agile Product and Portfolio Management – Crucial for Competitiveness

Jarno Vähäniitty

Success in today's software industry requires integrating long-term product and business planning with technology development, juggling the scarce development resources so that at each moment those activities that from a business perspective are the most important get attended to, as well as the combination of flexibility and control provided by modern, agile approaches to software development, such as Scrum. However, this is not easy, and to be compatible with agile software development, the enterprise level processes of product and portfolio management have to be understood in a new way.

In this chapter, we start off by briefly introducing the concepts of agile software development, product management and portfolio management (Sections 2.1-2.3). Then, we describe why making an agile software development process work together with the product and portfolio management processes as a harmonious whole is crucial for today's software business but also wrought with difficulties (Section 2.4). We round off the chapter in Section 2.5 by explaining how this book is meant to help you.

2.1 What is agile software development?

The term *agile software development* was coined in the year 2001 when the Agile Manifesto[6] was formulated (Hansson et al. 2006). Agile software development emphasizes building releasable software in short, fixed time periods and emphasizes flexibility, communication, collaboration and working software over processes, tools, documentation and following a pre-defined plan (Rico, Sayani & Sone 2009).

[6] http://agilemanifesto.org/

While there is no universally accepted definition of what agile software development entails (Kettunen & Laanti 2006), most approaches that are considered agile possess the following characteristics (Smith 2008):

- The development proceeds iteratively in loops of two to six weeks
- 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
- The (representative of the) customer is incorporated in the planning
- Small, close-knit cross-functional teams do the work

As the most common manifestation of agile software development, we have chosen the Scrum process framework to serve as the basis for most of the examples and guidelines presented in this book.

2.2 What is product management?

Product management deals with the planning, development and marketing of a product or products at all stages of the product's lifecycle, spanning from strategic to tactical activities (Ebert 2009). 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**. This 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 *engineering aspects* of software product management – on which we focus in this book – 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). The related core engineering processes are **portfolio management**, **product planning**, **release planning** and **requirements management** (Bekkers et al. 2010).

These processes, and the problems of fitting them with agile methods such as Scrum is returned to in more detail in Chapter 3.

2.3 What is portfolio management?

Portfolio management is a term that has many different meanings depending on the context; for example, a financial portfolio, a product/service portfolio, a portfolio of development projects, and so on.

In the literature on new product development, *portfolio management* (or *portfolio* management for short) refers to the decision-making process for updating and revising a business's product development portfolio, that is, the list of active and planned development activities that require the development resources' attention. In portfolio management, new projects are evaluated, selected and prioritized, existing projects may be accelerated, de-prioritized or killed, and resources are allocated to and reallocated within active projects (Cooper 2009)[7].

In this book we are mostly concerned with the *software development portfolio* (or *development portfolio*, or simply *portfolio* for short). By this we refer to the set of ongoing and upcoming *development activities* that require attention from the product development and/or technical resources.

A more thorough explanation of the concepts of portfolio and portfolio management along with illustrations as well as shortcomings of the existing literature from the perspective of understanding how to link portfolio management with agile methods follows in Chapter 3.

2.4 So, what is the problem?

"So really, what's the problem?" you may ask. For indeed, there are plenty of books on product management, new product development portfolio management, as well as on agile software development.

The problem is twofold:

First, based on a systematic review of the literature[8] we conducted in the autumn 2010, rather few authors so far deal with how product and portfolio management should be organized or even understood together with agile software development methods. However, some do, and we have attempted to synthesize those guidelines with our own findings in this book.

Second, while doing agile "right" is in principle simple, it is also extremely difficult, and for some people (as well as organizations!) requires a lot of unlearning to take place. As understanding how product and portfolio management can be made agile-compatible is not common knowledge, or even that well described in the latest literature, it is not a surprise that practitioners have a lot of problems in getting everything to work together. This difficulty is quite evident in many – if not most – organizations today that are striving towards adopting agile software development methods.

[7] Being a pragmatist, Cooper's concept of portfolio management actually includes both *product* portfolio management– as well as managing those activities that take up time from the development people – whether the latter are actual development projects or not (Cooper, Edgett & Kleinschmidt 2002)!

[8] As represented by both the Scopus database as well as the Amazon bookstore; for the review method, keywords and similar details, contact the authors

But before venturing further, Sections 2.4.1-2.4.3 briefly explain in more detail why solving the problem of linking agile with product and portfolio management is of crucial importance for companies that intend to be successful.

2.4.1 Most companies are hybrids

The majority of the companies in the software business, especially when examined over time, employ a 'hybrid' business model (Cusumano 2003). This means that besides offering products and striving to productize the technologies related to their key business idea(s), they on the side have to offer professional services and custom development projects – which may or may not be related to the company's products – to share risk and balance their cash flow (Artz et al. 2010, Cusumano 2004, Cusumano 2008).

This is especially true for small companies, most of which either do not wish or simply are not able to acquire significant external funding (Rönkkö et al. 2009). Indeed, out of the internally funded small companies, those who wish to grow profitably are hybrids out of necessity (Miettinen, Mazhelis & Luoma 2010, Wangenheim et al. 2006).

However, 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", and an improper balance between product development and servicing efforts has been referred to as "an easy way to ruin an otherwise good business" (Cusumano 2004).

2.4.2 The hybrid business model needs agile product and portfolio management

Hybrid software companies have to master software development and the enterprise level processes of *product* and *portfolio management*. This is illustrated in Figure 2.1 and further explained below.

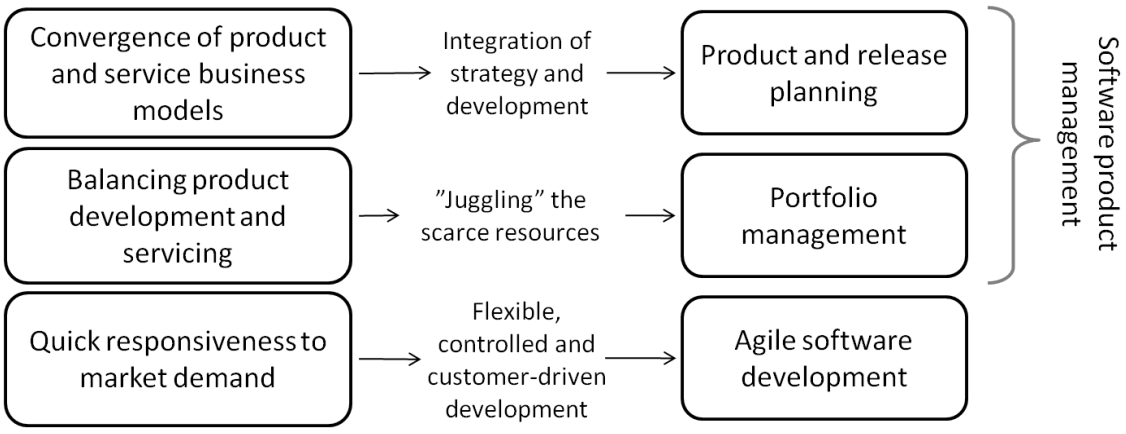


Figure 2.1: Demands for software companies' internal processes in today's software industry

First, the business environment is getting increasingly challenging. 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 to gain a significant position in global markets, does 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 ongoing convergence of product and service business models (Cusumano 2008, Rönkkö et al. 2009, Johnson 2010). This makes overcoming the challenge of integrating long-term business, product and release planning with technology development (Berry & Taggart 1998) increasingly crucial – even when the future and the available business opportunities seem to change so rapidly that long-term planning seems pointless (Doz & Kosonen 2008).

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 perform both product development and servicing[9], 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 as *portfolio management of new product development projects*, or simply portfolio management (Cooper, Edgett & Kleinschmidt 2002).

Third, development must possess the capability to quickly respond to emerging opportunities and market demand (Rönkkö et al. 2009, Cusumano et al. 2009). To make this possible, the employed development processes must be flexible, controlled and driven by customer needs (Takeuchi & 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). Agile software development also 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) with outsourced development teams (Sutherland et al. 2007) – not many such impressive results have been presented to date. Overall, current empirical evidence regarding agile methods' effectiveness is scarce and largely anecdotal (Dybå & Dingsøyr 2008), but the little that exists can be con-

[9] Here, *product development* 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 opportunities for innovation and needs for productization (Ruokonen 2008).

sidered encouraging (Cardozo et al. 2010, Syed-Abdullah, Holcombe & Gheorge 2006).

2.4.3 Agile is disconnected from product & portfolio management

Though some consultants may have a vision of it, in the existing literature the relationship between the product and portfolio management processes and agile software development is at best unclear. These challenges (see Figure 2.1) are illustrated in Figure 2.2, briefly explained below as well as further discussed in Chapter 3.

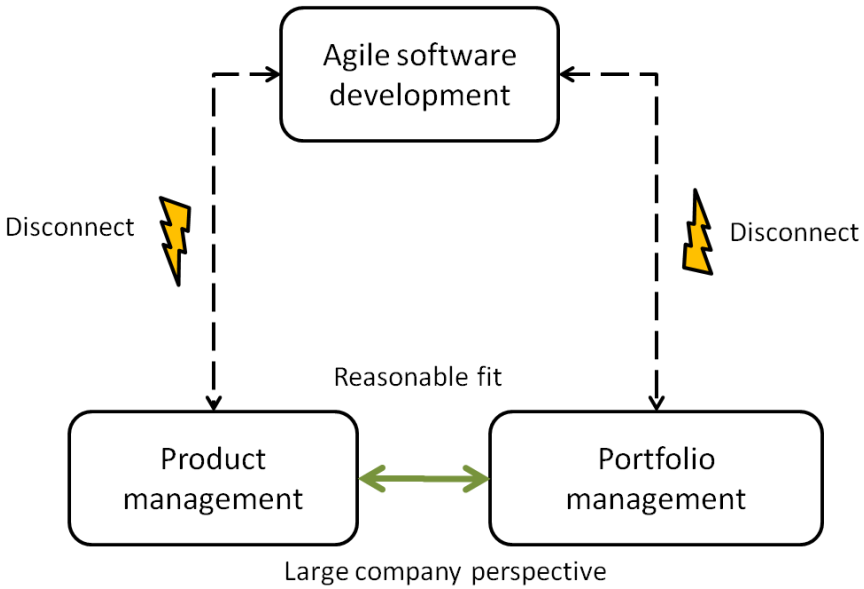


Figure 2.2: Literature on agile software development is disconnected from product and portfolio management literature

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 product and release planning (Valkenhoef et al. 2010) or portfolio management (Kettunen 2007), or indeed, with the running of an entire company (Kettunen & Laanti 2006), has not been discussed.

In recent research 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). Furthermore, from the perspective of small companies, it is problematic that 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). Such approaches do little to utilize the strengths of small companies such as flexibility, quick responsiveness and informal but direct communication structures (Wangenheim et al. 2006, Pino et al. 2010, Beattie & Fleck 2005).

2.5 How this book helps you

We have written this book with the intention of helping you tackle the challenge you are bound to face during your quest towards enterprise-level agility: integrating agile software development methods with product and portfolio management processes.

In the rest of Part I (*Introduction to time pacing and agile product and portfolio management*) we examine the product and portfolio management processes more closely and the **gap in the literature** regarding their linkage with agile software development methods (Chapter 3). If you are not yet convinced that there is a problem with the existing literature in describing the linkage – or, if you are a researcher – then this chapter should prove interesting.

In Part II (*Assessing the health of your portfolio management*) we present **the Portfolio Management Health Barometer** – a method for assessing whether your company needs to improve on its quest towards enterprise agility and how – as well as the theoretical underpinnings of the method. We also describe in detail how to use the method and its accompanying open source survey tool to conduct a *health barometer assessment* for your company – or for another company, should you be in the consulting business.

In Part III (*Framework and practices for agile product and portfolio management*) of this book we present our **framework of agile product and portfolio management** that describes how the key functions of the product and portfolio management processes should be understood in the context of agile development, along with practical examples from the companies we have worked with. 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 conducted as explicit projects. As we will see, such scenarios, though rarely addressed in the literature, may be fairly common in practice. Part III also examines some more challenging aspects of linking agile software development with product and portfolio management more closely, and presents practices that have proven useful, such as *joint release planning* for scaling up agile release planning (Chapter 10), *the agile requirements refinery* for organizing product management (Chapter 11) and using a *Kanban board* for managing a development portfolio (Chapter 12).

Part III closes with a summary of **the requirements for backlog management tool support** (Chapter 13) as posed by our framework for agile product and portfolio management. It also presents how some of these requirements have in practice been implemented in Agilefant (www.agilefant.org) – the leading open source tool for backlog management whose development is coordinated by the authors.

Chapter 3: The Gap in the Literature

Jarno Vähäniitty

This chapter examines the literature that can be considered as related to the problem of linking product and portfolio management with agile software development. When management aspects are concerned, the focus has in the software engineering literature traditionally been that of individual projects, thus neglecting much of the complexities in linking agile software development, long-term product and release planning and portfolio management. Literature on agile software development is no exception, as it until very recently has largely swept much of the complexities involved under the proverbial rug of the product owner role. Likewise, literature on software product management tends 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.

We start off with a closer examination of the concept of software product management, and identify those core processes that either make or break agile product management: product planning, release planning and requirements management (Section 3.1). Then we discuss these processes and the theoretical and consequent practical difficulties of applying existing work from the software product management literature to the context of agile software development (Section 3.2). The final section (3.3) of this chapter discusses portfolio management and explains why traditional phase-gate models used for implementing portfolio management can are problematic in the context of agile software development. We also explain the *levels of portfolio management* crucial for agile portfolio management, and define more closely the concept of *development portfolio management* as the most important of these levels.

3.1 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 from strategic to tactical activities (Kahn, Castellion & Griffin 2005). In line with this, **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). It consists 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).

At least in Finnish software companies, product management can be a complicated and somewhat confusing matter based on our experience as well as the experience of others (Sahlman & Haapasalo 2009, Saastamoinen & Tukiainen 2004). For example, long-term business, product, and release planning are often carried out implicitly and without being properly integrated to day-to-day operations. However, it is known that successful high-tech companies, even small ones, do practice explicit planning to direct their development efforts (Berry 1998), and the planning process and its link to development become more sophisticated as the companies grow (Berry & Taggart 1998).

In this book we focus on the so-called core processes of software product management as defined by the software product management competence model (Bekkers et al. 2010, Weerd et al. 2006). The framework is illustrated in Figure 3.1 below.

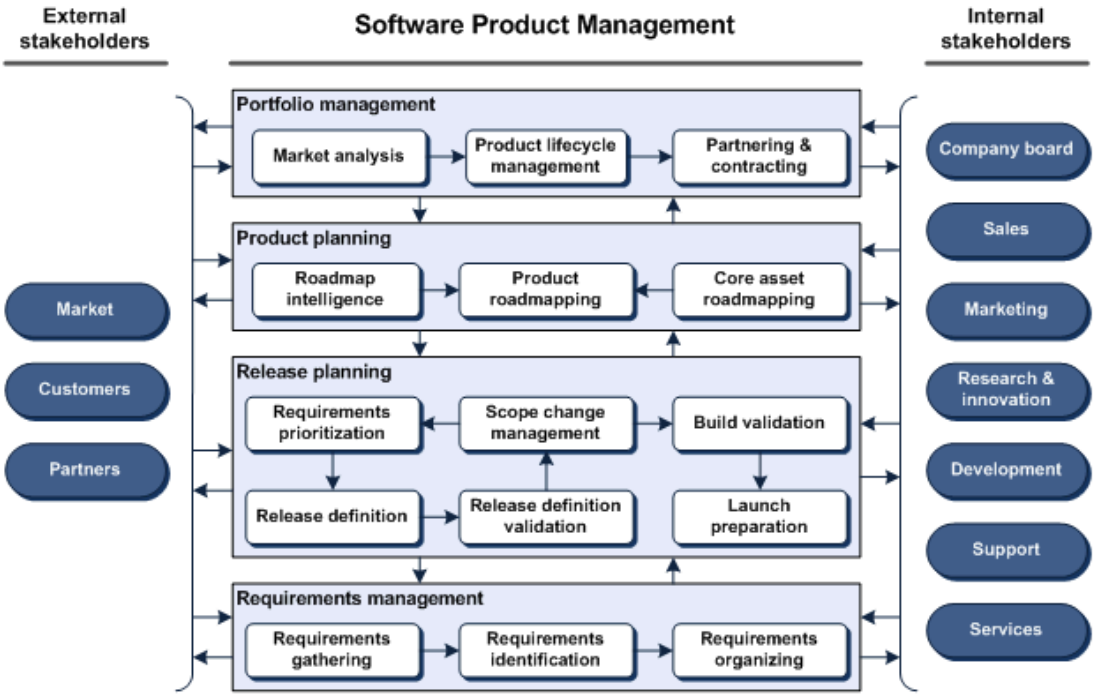


Figure 3.1: The Software Product Management Competence Model (Bekkers et al. 2010)

The following excerpts describe the core processes:

Portfolio management *concerns the strategic information gathering and decision making across the entire product portfolio* (Bekkers et al. 2010). *It entails making decisions about the set of existing and new products based on the market trends and the product development strategy, and establishing partnerships and contracts* (Weerd et al. 2006). *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 a product's lifecycle 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.* (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 support information needed in the creation of the product roadmap.* Product roadmapping *deals with the actual creation of the product roadmap.* Core asset roadmapping *concerns the planning of the development of core assets (components that are shared by multiple products).* (Bekkers et al. 2010)

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 the preparations for the implementation of the release itself are addressed.* (Bekkers et al. 2010)

Requirements management *comprises the continuous management of requirements that are not (yet) part of a release and consists of three focus areas.* Requirements gathering *concerns the acquisition of requirements from both internal and external stakeholders.* Requirements identification *identifies the actual Product Requirements by rewriting the Market Requirements to understandable Product Requirements, and connecting requirements that de-*

scribe similar functionality. Requirements organizing *structures the requirements throughout their entire lifecycle based on shared aspects, and describes the dependencies between Product Requirements.* (Bekkers et al. 2010) *Requirements management is more or less a continuous activity, while product and release plans are revised at discrete points in time* (Kittlaus & Clough 2009).

Based on these definitions, we make two observations crucial for linking agile software development with product and portfolio management:

First, the body of literature on software product management views portfolio management as a very high-level activity dealing with coarse-grained allocation of capital (Kittlaus & Clough 2009), in other words, as *product portfolio management*. This leaves out several perspectives that are essential for getting agile software development to work, for example, that of managing multiple simultaneous activities such as ongoing development projects or other efforts that take up the development personnel's time. We will delve into this in more detail in Section 3.3 as well as return to it in Part III of this book.

Second, it is obvious that the way in which product planning, release planning and requirements management are understood and organized determines whether product management is "agile compatible" or not – and thus, can be the obstacle for achieving enterprise-level agility.

But, the problem is that existing literature on software product management, or software engineering, does very little in the line of providing guidelines how this should be done in practice in the context of agile software development. This is discussed in Section 3.2.

3.2 Key processes for agile product management: product and release planning

In this section we summarize existing literature and knowledge on long-term product and release planning, in other words, those two core processes of the software product management competence model called product planning and release planning (Bekkers et al. 2010).

3.2.1 Product planning a.k.a. 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). In this book we use it as a synonym for the concept of product planning as defined in the software product management competence model (Bekkers et al. 2010) discussed in Section 3.1.

The process of roadmapping should identify, evaluate and select strategic alternatives for achieving desired objectives (Kostoff & Schaller 2001). The resulting

roadmaps summarize and communicate the results of key business decisions (DeGregorio 2000). Thus, the roadmaps' implementability is at least as important as their possible strategic value (Kostoff & Schaller 2001). The problem is, that there is little research literature on software product roadmapping (Fleury et al. 2006) – and none on how to do, view or even understand roadmapping in the context of agile software development.

Some of the most relevant software (product) roadmapping research has been conducted by Lehtola et al. (Lehtola et al. 2009, Lehtola & Kauppinen 2006, Lehtola, Kauppinen & Vähäniitty 2007, Lehtola, Kauppinen & Kujala 2005), although the focus taken is not specific to agile software development. In the mentioned publications Lehtola et al. make several observations about the state of practice of roadmapping in medium-sized software companies based on their case studies. Project-wise requirements engineering seems "insufficient to provide a holistic long-term view with different stakeholders". The preparation of roadmaps seems to be disconnected from development and is the responsibility of the product managers, who many times are relatively distant to the daily work in the actual development projects. As a result, business goals and needs are "hard to trace to development" and there is "no clear link between business and development decisions". Also, allocating development resources according to the plans made in the roadmapping phase is very difficult in practice, often involving competition for shared resources – which in turn has been noted to lead to local optimization (Larman & Vodde 2008).

To counter the found challenges, Lehtola et al. (2009) propose five practices to help link development and business decision-making:

1. explicating the planning levels and time horizons,
2. separating the planning of products' business goals from R&D resource allocation,
3. open-ended planning with a pre-defined cadence,
4. emphasizing whole-product thinking, and
5. making product planning visible.

Done properly, roadmapping would strengthen the link between product management and agile software development. However, current research indicates that practices for roadmapping seem scarce and undeveloped both in practice as well as in terms of research efforts. Thus, what we would need now is first, to understand what roadmapping actually means in the context of agile software development. Second, we need a practical example of a roadmapping approach suitable for organizations developing software and related services that would follow the above guidelines and practices – as well as the possible desire of an organization to apply agile methods to manage at least some of their develop-

ment activities. To respond to these needs, Part III of this book presents our approach to agile roadmapping.

3.2.2 Release planning

Release planning (also known as "product release planning" and "strategic release planning") is concerned with selection and assignment of requirements in one or several sequences of releases in a way in which important business, technical and resource constraints are fulfilled (Svahnberg et al. 2010). Release planning has attracted attention among software engineering researchers. However, the existing models (Akker et al. 2008, Chatzipetrou et al. 2010, Ngo-The & Ruhe 2009, Ruhe 2010) treat release planning essentially as an optimization problem (Kittlaus & Clough 2009).

According to a recent review (Svahnberg et al. 2010), the existing models are designed for a situation where there is a single product/service offering with a set of possible features to be selected from. These features are assumed to have been elaborated to the degree that their development cost and business value can be reasonably estimated. Also, it is assumed that a group of relevant stakeholders is readily available to familiarize themselves with the requirements and vote on them.

Usually, one or more of the listed assumptions does not hold in practice (Svahnberg et al. 2010, Lehtola 2006). For example, the degree of up-front requirements elaboration needed by the approaches is often not feasible or even desirable (Larman & Vodde 2010, Poppendieck & Poppendieck 2009). Also, in practice requirements are not prioritized as a one-off activity, but in multiple phases of development, with each phase involving different kinds of decision-making (Lehtola 2006). Furthermore, there are often requirements from more than a single product/service offering for the development staff to work on (Rothman 2007, Dobson 1999, Rothman 2009).

Overall, existing systematic algorithmic approaches for planning the future development steps of a particular product/service offering often seem to have unfortunately little applicability to the actual decision-making problem faced by practitioners (Ivarsson & Gorschek 2009). Thus, it is hardly a surprise that most approaches to release planning have not been validated in an industrial setting (Svahnberg et al. 2010).

Rather than further devising models for "optimizing" the contents of upcoming releases, we argue that it should first be understood how the roadmapping and release planning processes actually manifest themselves in agile software development. This is further explored and explained in Part III of this book.

3.2.3 Software Product Management literature is Disconnected from Agile

Literature on software product management tends to view development as an activity that can be planned for and then carried out according to the plan – although it is sometimes recognized that "slippages caused by the development" (Kittlaus & Clough 2009) are possible. For example, in Figure 3.1 (page 39) the effects of scope changes from development extend to requirements management, product roadmapping and portfolio management only indirectly. In this section we present several excerpts that seem to support the notion that most of the literature on software product management has been written with a sequential, waterfall-like software development life cycle in mind. Thus, it does little to help link product management with agile software development.

In one of the earliest books on software product management (Condon 2002), there are several pages dedicated to discussing Extreme Programming (Beck 2000). The following excerpt hints that the basic concepts behind agile software development may be foreign to a software product management worldview:

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 not have 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)

As another example, the following excerpts from Kittlaus & Clough (2009) concerning the documentation of requirements seem to represent a plan-driven approach to software development:

Requirements management is a documentation-intensive task. Each requirement must be individually documented. Then all requirements from all the diverse sources are combined into 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 re-

quirements management must allow changes during the development process in a tightly managed manner. (p. 92)

As another example, the following paragraph in (Kittlaus & Clough 2009) contains the book's only discussion regarding 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. 95)

The above excerpt seems to exhibit what Larman and Vodde (2010) refer to as a *competitive contract game* that "inhibits building the right thing and building the thing right".

As the final example from the gray literature on software product management, a recent book (Ferrari 2008) – which contains no discussion of agile methods – has the following paragraph concerning requirement priority changes during development. While the author recognizes the counter-productiveness of trashing, the underlying assumption seems to be that the requirements are realized in sub-projects that have detailed task-level plans:

Once priorities have been set and a product plan is in production, avoid priority changes. After the development organization has started working on something, making a change that stops that work will cause frustration and performance losses due to the deceleration and re-acceleration people go through in their tasks as they are moved from one project to another. (p. 131)

The above thinking seems in stark contrast to the continuous planning and just-in-time elaboration (Shalloway, Beaver & Trott 2009) advocated by the agile movement.

Based on the above examples, both grey as well as research literature on software product management would seem to be disconnected from agile software development. The following two excerpts from grey literature on agile software development seem to address what may underlie this problem:

To those in the software product management community, the role of product owner in agile methods looks like a new version of "barbarians at the gate" - the development community attempting to extend their fingers into areas where they lack distinctive competence, appropriate background, and training. To the software product management community, Scrum is a process,

built by developers, for developers, but who says that processes and those new roles will now be used to drive product and market policy? In their view, that's what product managers do now, and have always done, so why would they let the software development types assume their responsibilities. (Leffingwell forthcoming 2011, Chapter 11 Role of the product owner)

Officially, many product developers 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 [must] 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, the average 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. (Reinertsen 2009, pp. 1-2)

From this background, it is understandable that the work on connecting software product management with agile software development is only at its very beginning. Some steps are being taken(Vlaanderen et al. 2009, Mohan, Ramesh & Sugumaran 2010), but the work on software product management has not on the conceptual level yet been properly integrated with the recently proposed approaches and frameworks in the gray literature (Shalloway, Beaver & Trott 2009, Larman & Vodde 2008, Rothman 2007, Rothman 2009, Leffingwell forthcoming 2011, Schwaber 2007, Pichler 2010, Krebs 2008, Schiel 2009) that attempt to extend agile into the domain of software product management. Also, no experiences from adopting or applying such integrated frameworks have yet been reported in the software product management literature. We will return to both of these topics in Part III of this book.

3.3 Portfolio management

In the following subsections we describe why traditional phase-gate models used for implementing portfolio management are easily seen as fundamentally incompatible with agile principles (Section 3.3.1). Then, we describe the six levels of portfolio management that are essential for a proper understanding of

the concept in the context of agile software development (Section 3.3.2). We continue by explaining perhaps the most crucial of these levels from the perspective of getting portfolio management to work in an agile-compatible fashion: *development portfolio management* (Section 3.3.3), and provide an example of what a typical development portfolio looks like in a small organization that develops software (Section 3.3.4).

3.3.1 Agile vs. phase-gated product development models

Portfolio management has been written about extensively in literature on managing new product development, and potential for cross-domain knowledge is said to exist (Nambisan & Wilemon 2000). 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 there is manufacturing involved in getting the products to the market (Martinsuo 2001). Setting up proper portfolio management has been recognized as challenging even for the best of organizations (O'Connor 2004).

Portfolio management in the context of agile software development has not been discussed. In large product development organizations, there is typically a portfolio of concurrent product development projects to be managed (Kettunen 2007), and instead of abandoning traditional Phase-Gate models for product and portfolio management, these should somehow be combined with agile software development (Dybå & Dingsøyr 2008). 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 end result is quite similar to that of larger companies.

Advocates of agile methods see much of the literature on managing new product development as fundamentally incompatible with agile software development. Perhaps this is because of the tendency of new product development literature to view and define development as a separate “phase” in the cycle of realizing an idea into an actual working solution (Krebs 2008).

Also, the Stage-Gate™ (Cooper 2009) – or at least how it is commonly understood – is by some seen as incompatible with the basic principles of agile development (Larman & Vodde 2008). There have been case studies on integrating phase-gate type models and software development life-cycle models (Wallin, Ekdahl & Larsson 2002), even including agile methods (Karlström & Runeson 2006). The latter study found, among other things, problems in adapting corporate practices for project planning to accommodate for “agile micro planning” in combination with the “normal” macro project planning. We see this as a sign of disconnect between the “normal” plans and the reality of the agile teams in question.

Overall, the work on software product management and portfolio management seems so far to be largely disconnected from agile software development. In Part III we combine our experience and findings with what literature on agile software development has to offer in terms of product and portfolio management.

3.3.2 Levels of portfolio management

We have already mentioned in Chapter 1 that the term 'portfolio management' has many meanings in different contexts. But this is true even within the context of managing software development. Just as the software product management competence model implicitly referred to *product portfolio management* as 'portfolio management', different experts even within the agile community are using the term to refer to different kinds of decision-making.

Using the Cycles of Control framework (Rautiainen et al. 2006) as a basis, managing software development can be viewed as consisting of multiple nested planning horizons. This idea and its link to portfolio management is illustrated in Figure 3.2 and explained below.

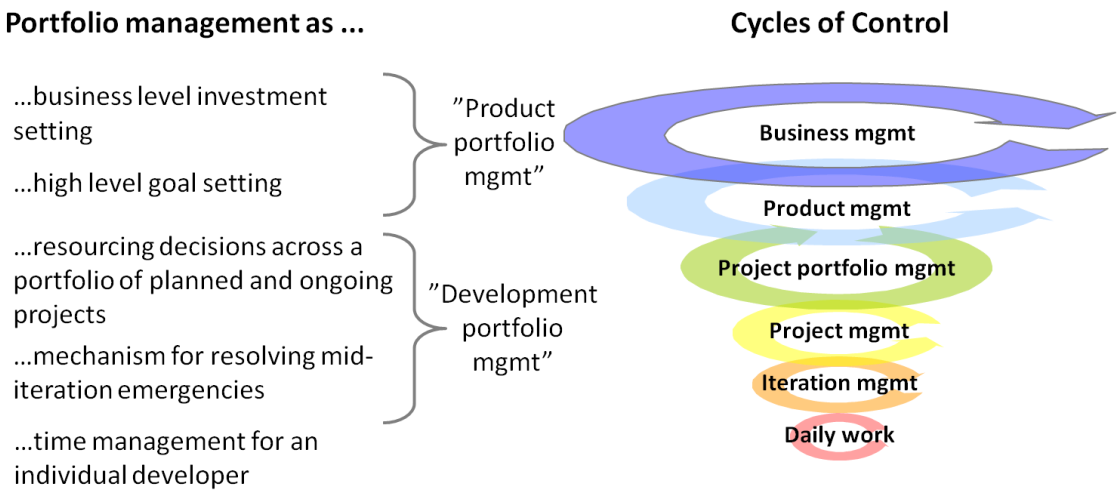


Figure 3.2: Levels of portfolio management in agile software development

As depicted in Figure 3.2, there is a different kind of "portfolio" to manage on different planning horizons. In *business 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. As we have already discussed, this portfolio management is *product portfolio management*. One planning horizon down, when managing a single product/service offering (the *product management* cycle in Figure 3.2), decisions are made regarding the epics and features to be developed in the current and upcoming releases. Going further down the cycles, in *managing a release project*, there is a portfolio of features that are planned to be 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. An individ-

ual 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 as part of his personal portfolio.

The levels of agile portfolio management are further discussed in Part III in Section 8.1. But next, we elaborate on development portfolio management, as getting it to work is in the heart of linking product and portfolio management with agile software development.

3.3.3 Development portfolio management

By the software development portfolio (or *development portfolio*, or simply *portfolio* for short) we refer to the set of ongoing and upcoming *development activities* that require attention from the product development and/or technical resources; see Figure 3.3 below.

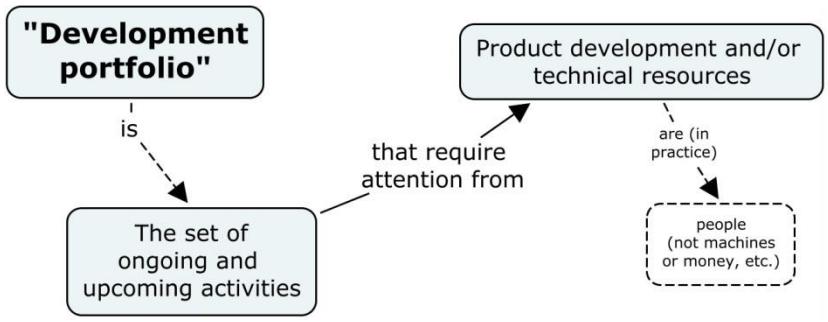


Figure 3.3: Software Development Portfolio

Examples of different types of *development activities* – or just *activities* for short – are release-based development, customer-specific development, maintenance, deliveries, customer service and consulting (see Figure 3.4 below).

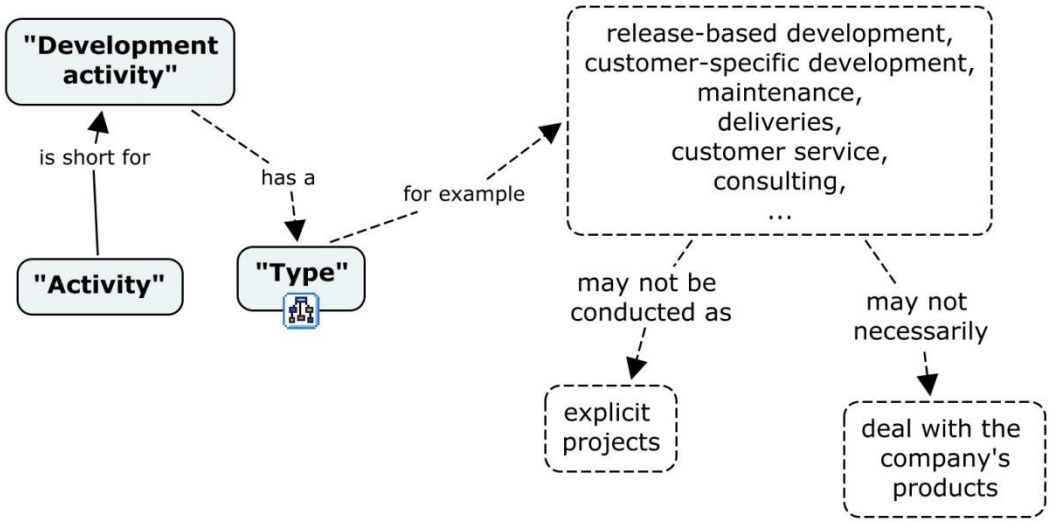


Figure 3.4: Development activities

Software development portfolio management (or *portfolio management* for short) is the decision-making process for updating and revising a business's development portfolio, that is, the list of active and planned development activities

that require the development resources' attention (see Figure 3.5). In portfolio management, new projects are evaluated, selected and prioritized, existing projects may be accelerated, de-prioritized or killed, and resources are allocated to and reallocated within active projects (Cooper 2009).

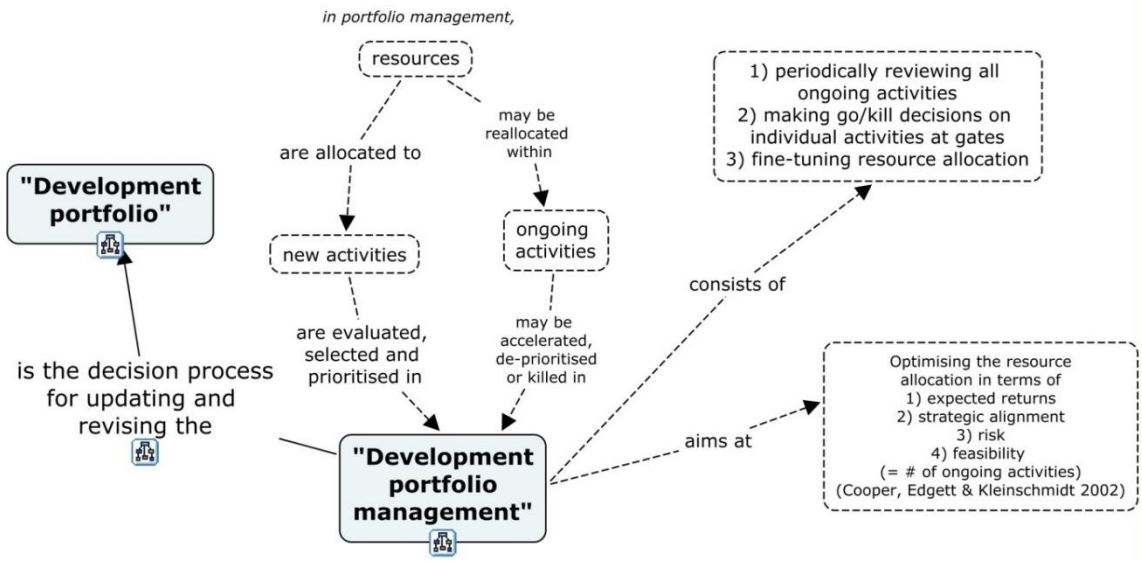


Figure 3.5: Development portfolio management

3.3.4 An example development portfolio

Figure 3.6 below illustrates an example development portfolio from HardSoft.

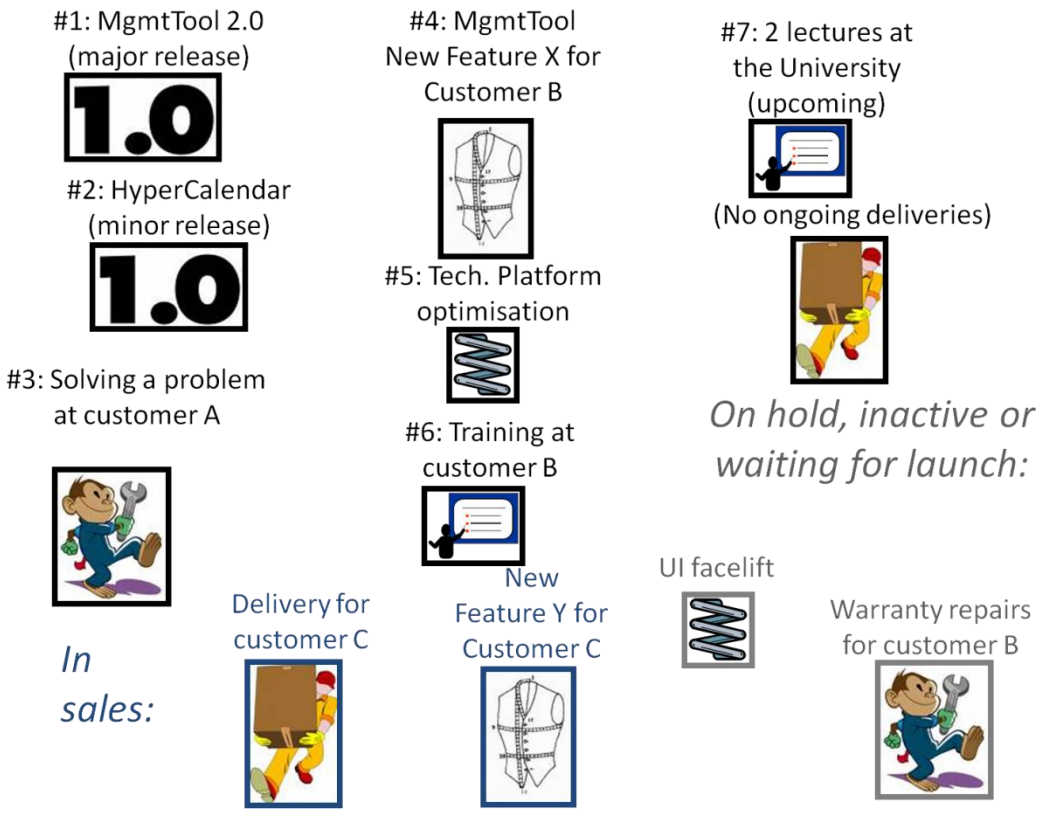


Figure 3.6: A snapshot of the development portfolio at HardSoft

> At HardSoft (see Figure 3.6 above) there currently are two ongoing release-based product development projects (activities *#1-MgmtTool 2.0* and *#2-HyperCalendar*) and some people are involved in maintenance work (*#3 Solving a problem at customer A*). In addition, new functionality for one of the major products of the company is being developed in a project for a key customer who has in a separate contract agreed to pay for it (*#4 MgmtTool New Feature X for Customer B*). Some attention is being spent on optimizing the platform used by both of the major products of the company (*#5 Tech. platform optimization*). A couple of people are to conduct two training days to get a key customer started on using the MgmtTool product (*#6 Training at Customer B*), and are preparing for these. To get the company some visibility, the same guys are also to hold lectures at the local technical university (*#7*) based on largely the same material they are preparing for the training days. While – luckily – there are no ongoing delivery projects, a sale consisting of both delivery and the development of new functionality required by the customer (*Delivery* and *New Feature Y for customer C)* is likely to be closed in the near future. There has been talk about remaking the user interface with a newer technology (*UI Facelift)*, and it is unclear whether somebody is – or should be – working on it. Finally, the warranty period in which "all found bugs are promised to be dealt with within a week" (which means either producing a patch release or inventing and explaining a feasible work-around) is still going on for the key customer mentioned earlier (*Warranty repairs for customer B*), and at any moment, some work might arise from that direction.

The above example is in our experience rather typical, and can be encountered in many (if not most) organizations that develop software - whether, small, medium-sized or larger. Note that while the depicted situation may seem chaotic or difficult to manage at best, the organization in question is actually already doing quite well in terms of *structuring* its development portfolio – something that we will return to in Part III. This is because it has actually defined what activities are *ongoing* (or *in sales* or *on hold*), as well as the distinct *types of development activities* (to be explained in Part III).

PART II: ASSESSING THE HEALTH OF YOUR PORTFOLIO MANAGEMENT

Before you start an initiative to introduce or improve a development portfolio process in your organization you should have a solid picture of the current state of affairs. This part describes a method you can use to assess the health of your portfolio management processes. We call the method development portfolio health barometer study (HB). HB has been developed over the years by the researchers of SoberIT's Software Process Research Group (SPRG). The study has over a period of five years been performed from two to four times in eight different organizations that vary in business area and size. In each and every of those organizations we have uncovered things that the organization has found important and interesting. We start this part by describing the theoretical background behind the health barometer study in Chapter 4: The Portfolio Management Health Barometer. In Chapter 5: Performing a Portfolio Management Health Barometer Study we give concrete guidance on how you can perform a health barometer study in your organization. In addition to the theoretical background and the study method, we have also developed a tool to assist in HB. The use of this tool is described in Chapter 6: The Health Barometer Tool.

Chapter 4: The Portfolio Management Health Barometer

Jarno Vähäniitty

One of the best ways to assess your current situation in terms of product and portfolio management and their (lack of) connection with the development is to examine the health of your development portfolio management process. The Portfolio Management Health Barometer helps you assess whether your organization needs improvement, and if so, where you should start the improvement work. Section 4.1 explains in more detail the Health Barometer and the key ideas behind it, while sections 4.2-4.4 describe the actual issues measured by the Health Barometer as well as the theoretical underpinnings of why measuring these issues is important.

4.1 Examine your development portfolio management to find out where you stand

Software engineering literature is ripe with different approaches for assessing your development processes. Some of them aim towards certification and are based on a standard or the CMMI framework, while others are more interested in, e.g., trying to sort out how agile your development teams really are.

But, if we leave aside asking from your customers, we have found out that perhaps the best way to get an understanding of your real capability in the area of producing working good-enough solutions with real value is to examine the health of your development portfolio management process. This is because of its crucial role in enacting your strategy via resource allocation, as well as providing feedback from development to the business and strategy people.

Towards this end, we have developed the Portfolio Management Health Barometer. The Portfolio Management Health Barometer (Health Barometer or HB for short) is a structured and systematic way of assessing the adequateness of the current practices for development portfolio management over time.

The Portfolio Management Health Barometer is conducted for an organization as a study containing a questionnaire to be answered and in-depth interviews. The minimum needed for a study, in principle, is a single person to answer the

questionnaire, but that rarely produces quality results. The largest studies we have conducted consisted of having some 30 people answering the questionnaire and a third of them interviewed, which provided very useful results. Detailed instructions for conducting Health Barometer studies and on using the accompanying open source survey tool follow in Chapter 5 and Chapter 6.

The Health Barometer is based upon the fact that portfolio management decisions are always 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 sufficiently simple to be managed for each project or even without formal project management. For example, if the ongoing activities have no resource- or deliverable dependence, explicit portfolio management may not be needed. To assess whether an organization is actually suffering from the lack of explicit portfolio management, we need to know what symptoms occur in conjunction with inadequate portfolio management. If an organization exhibits many or most of such symptoms but does not intentionally or explicitly practice portfolio management, it is reasonable to propose that explicit portfolio management could be beneficial.

The most important function of the Portfolio Management Health Barometer is to record how the development portfolio management is being managed along with the health of the potential problem areas as perceived by the participants, and use these as a baseline for future HB rounds to measure the effectiveness of improvement efforts. However, it serves several other purposes as well (see Figure 4.1 below).

As illustrated in Figure 4.1, conducting a Health Barometer study has shown to raise the participants' (or subjects') awareness of the development portfolio management process (or the lack of it), all the issues that may be affected by it, as well as what factors increase the need for an explicit portfolio management process. From a research standpoint, the HB rounds we have conducted have provided us valuable data on how people of different roles and seniority view these issues[10].

[10] These results have not yet been published and thus are out of the scope of this book.

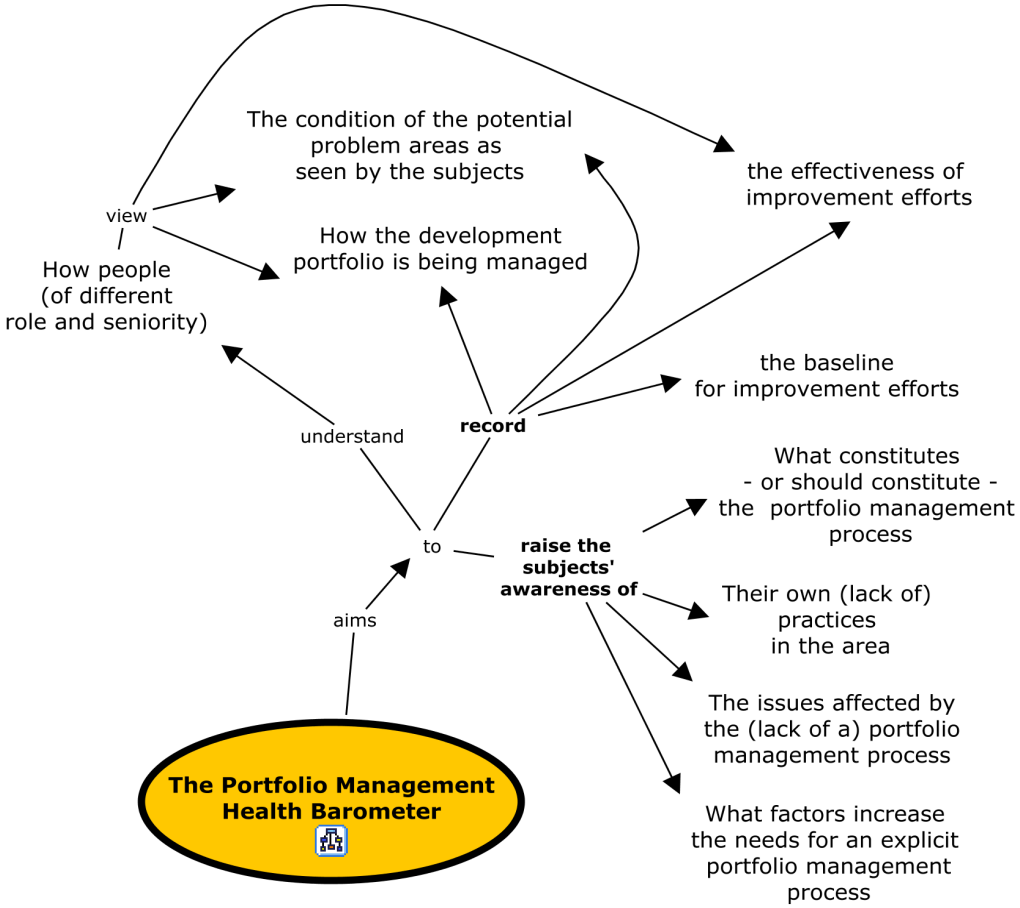


Figure 4.1: The Portfolio Management Health Barometer serves many purposes

Content-wise, the Health Barometer consists of four main sections: demographics, hereditary factors, lifestyle, and symptoms (see Figure 4.2 below).

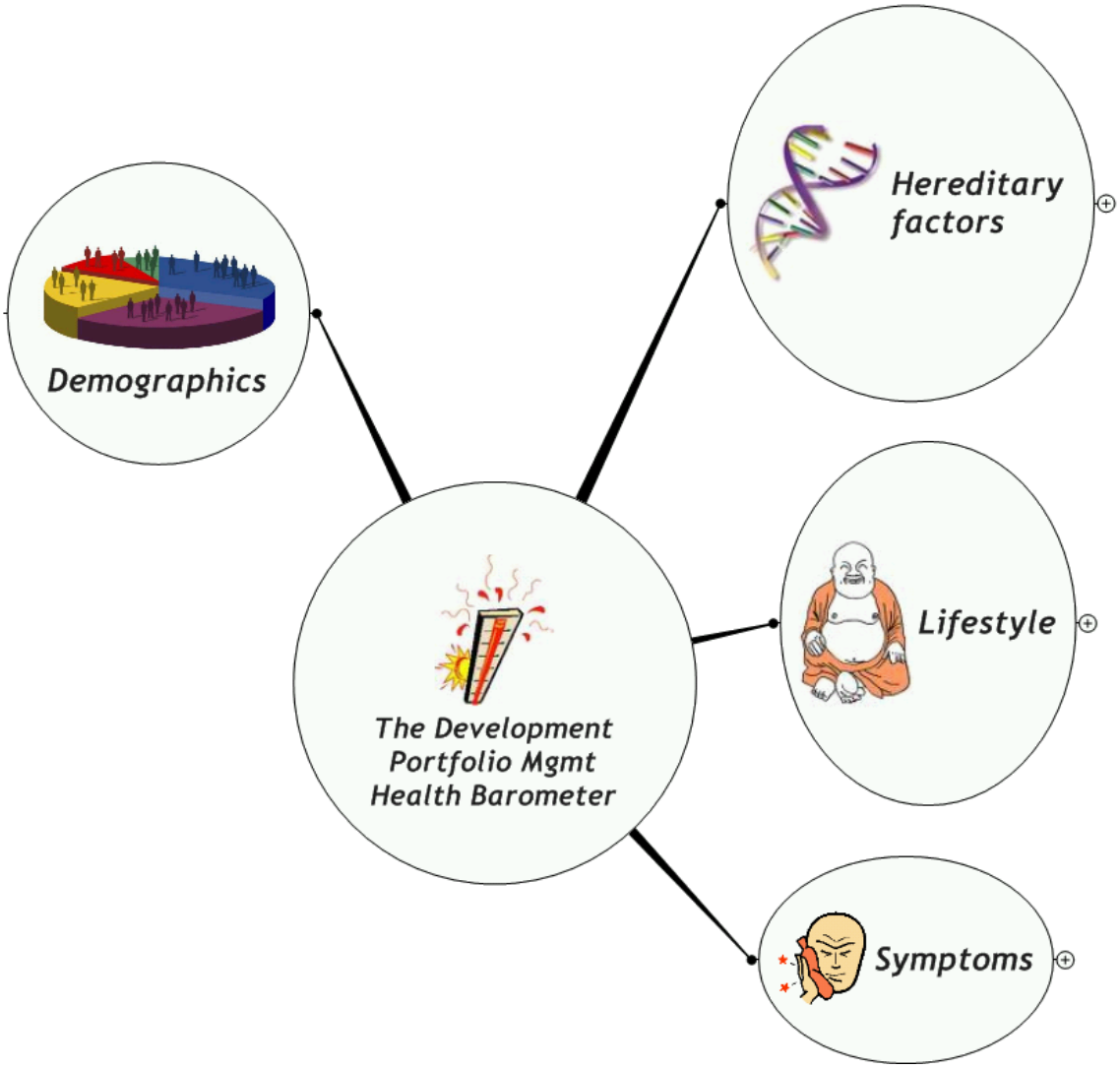


Figure 4.2: The Health Barometer measures demographics, hereditary factors, lifestyle, and symptoms

D*emographics* contain questions on the respondent's responsibilities, team, where their work place is physically located, duration of employment, and the profitability of the company. While the demographics section does not contain questions about the size of the organization that is being assessed – mainly because we don't want to bother the respondents with unnecessary questions that are better answered via other mechanisms – you should record this information!

By *hereditary factors*[11] we refer to organizational attributes that are outside the domain of development portfolio management, but increase the need for more rigorous development portfolio management if symptoms are to be avoided; for example, appropriateness of organization structure, reward systems, organization size, and business model. *Lifestyle* refers to how well the development portfolio is structured, and the actual processes and practices used for development

[11] Note, that while changing hereditary factors may be difficult or painful, it is by no means impossible –think of epigenetics…

portfolio management. By *symptoms* we refer to problems that in the literature have been associated with inadequate portfolio management (Vähäniitty, Rautiainen & Lassenius 2010), for example, late decision-making, inefficient resourcing and lack of focus.

With the exception of demographics, the specific contents of the main sections of the Health Barometer are explained in more detail in Sections 4.2-4.4 below.

4.2 Hereditary factors

The hereditary factors measured by the health barometer are illustrated in Figure 4.3 and explained in Sections 4.2.1-4.2.7 below.

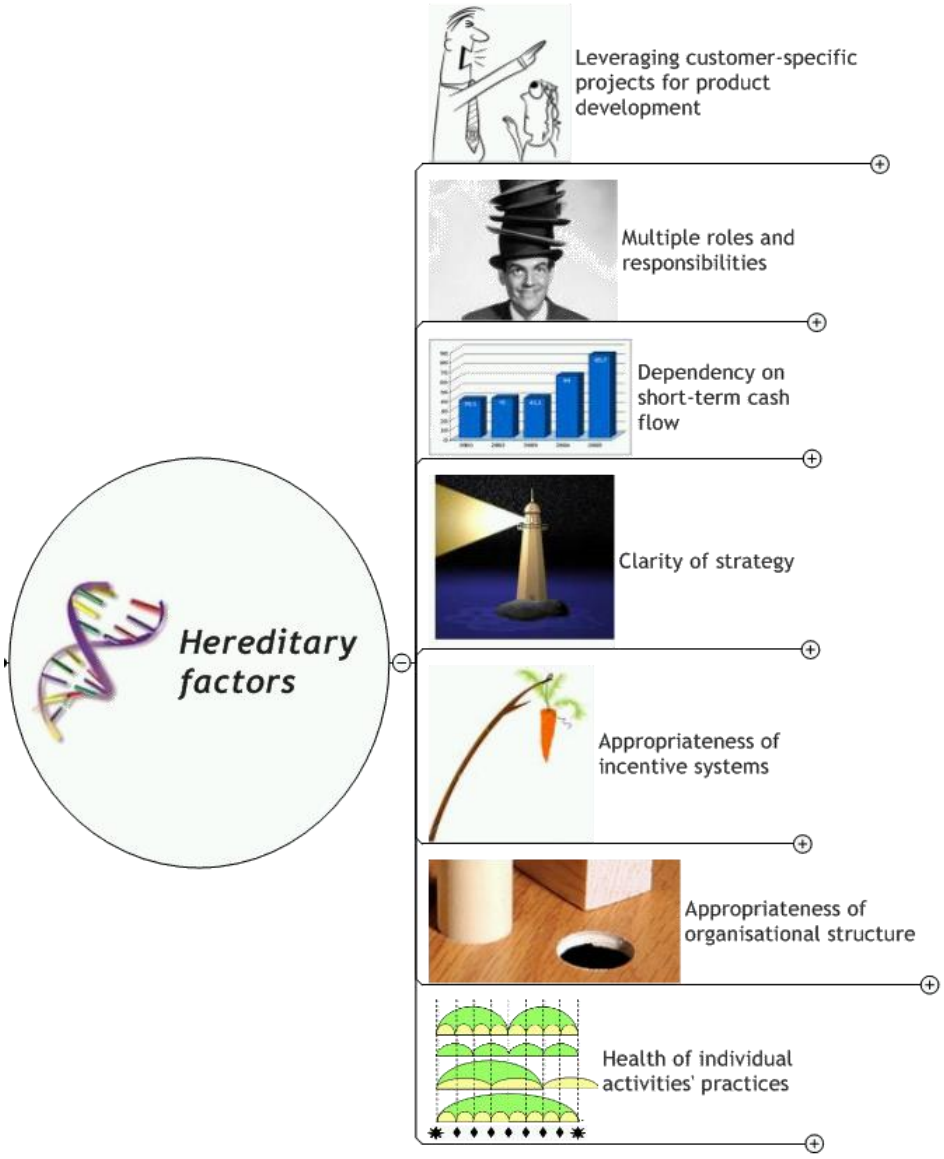


Figure 4.3: Hereditary factors measured by the Health Barometer

Note that some of the hereditary factors are such that having them is not 'good' or 'bad' as such. For example, small companies often have multiple roles and responsibilities per person (especially for the key people), which is the way it

should be. In the case of hereditary factors, a 'bad' score (see Section 5.5.1) simply means that on the average, leading a healthier lifestyle (that is, having a more rigorously structured development portfolio and an explicit portfolio management process) is likely to be beneficial for overall success.

4.2.1 Leveraging customer-specific work for product development

The statement for leveraging customer-specific work for product development is:

> New products or features are developed in customer-specific projects

Often, especially in small companies, new products and/or features are developed for a specific customer at first in order to share risk and generate revenue. Note that while the statement refers to 'projects' for the sake of clarity, it may be that only some of customer-specific work that produces useful output from the perspective of the offering(s) are managed as explicit projects. This further increases the need for explicit portfolio management.

4.2.2 Multiple roles and responsibilities

In the companies we have seen, the development people often have been working on many activities besides software development (for example, sales support, maintenance, deliveries, customer service, and consulting). The statement for assessing this is:

> Most of our development people have a broad work profile (e.g. they participate in many of the following: product development, customer projects, project management, sales / sales support, customer support, consulting, deliveries, training, etc.)

The more people have multiple roles and responsibilities, the more there is need for rigor in development portfolio management (Vähäniitty, Rautiainen & Lassenius 2010). This is partly because of the lost focus resulting from trashing between roles and activities. When this factor is combined with leveraging customer-specific projects for product development (see Section 4.2.1 above), rigor is truly needed. This is because the multiple roles of the employees are many times inherently conflicting (Vähäniitty & Rautiainen 2005). For example, the product manager at one of our case companies simultaneously acted as the manager of a certain customer-specific development project and recognized himself as biased toward accepting requests from his own customer with a less thorough consideration for the overall direction to which the product should head.

4.2.3 Dependency on short-term cash flow

The majority of software companies either are not able to acquire or simply do not wish significant external funding (Rönkkö et al. 2009), leaving them more

or less dependent on short-term cash flow. This factor is measured with the following statement:

> A downswing in cash flow is quickly reflected in the ability to pay salaries

While long-term product and business goals should set the framework for taking action, generating short-term cash flow and customer satisfaction cannot be neglected. Without rigorous portfolio management to maintain the delicate balance, significant amounts of effort can be spent on activities that ultimately are less important.

Also, many times at least some of the services a software company offers are not related to the products offered. For example, at our fictional example company HardSoft (that is based on a combination of experiences from real companies), some developers were performing management consulting, and a significant percentage of the entire development staff was contracted to other companies for longer-term software development projects. This, again, increases the need for explicit portfolio management.

Dependency on short-term cash flow, together with *Leveraging customer-specific work for product development* (Section 4.2.1) and having *Multiple roles and responsibilities* per person (Section 4.2.2) are in our experience the three most important factors that cause even small companies to have a need for explicit portfolio management (Vähäniitty, Rautiainen & Lassenius 2010).

4.2.4 Clarity of strategy

Development portfolio management, whether explicit or implicit, is the main mechanism for enacting strategy. Clarity of strategy is measured with the following statements:

> Strategy and long-term plans are clearly defined

> Strategy and long-term plans are clearly communicated

While even rigorous portfolio management cannot really function based on an unclear strategy, with implicit portfolio management the less-than-optimal situation of having an unclear strategy goes unnoticed much longer. Also, if the strategy has not been clearly communicated, it becomes harder for the development people to make the right decisions on their own and without escalating decisions upwards, which again increases the need for effective portfolio management.

4.2.5 Appropriateness of incentive systems

The incentives of the managers, sales people or even developers may be tied to their performance in a dysfunctional way from the perspective of effectively

managing multiple activities as a portfolio. This is measured with the following statement:

> Developers, project managers, sales, or senior managers are evaluated and rewarded in ways that are harmful to the whole

Over the last couple of decades, the financial community has become increasingly preoccupied with short-term success and a desire for fast profits[12]. This can be seen in the way senior people are often measured, and consequently, many senior managers have become "speed demons", placing far too much emphasis on accelerated time-to-market and cycle-time reduction. (Cooper & Edgett 2003)

To maintain balance, middle management (and in turn, developers) may attempt to procure the maximum time possible for projects and tasks, because their reporting and reward systems in turn may evaluate success according to how well the estimates are met. (Cerveny & Galup 2002, Dooley, Lupton & O'Sullivan 2005). And, despite that the staff already seems occupied at all times, management continues to take on additional work, as it does not wish to see an opportunity to make money slip away (Payne 1995).

Personnel reward systems in most organizations may also be divisive rather than integrative. Rewards based upon functional rather than organizational goals[13] easily work against the whole rather than for it (Payne 1995). For example, a department may be reimbursed for engineering hours spent on contracted projects, while non-project time, such as meetings, improvement activities, education, and slack time are accounted as costs. In this situation, the financial incentive is to spend as many engineering hours as possible on each singular project and there are little financial incentives for productivity improvements (Engwall & Jerbrant 2003).

4.2.6 Appropriateness of organizational structure

If the organizational structure is inappropriate, product development (or vice versa, functional responsibilities) may suffer (Cooper, Edgett & Kleinschmidt 2000). The health of the organizational structure is measured with the following statement:

> Our organizational structure supports our current operations

[12] While the inherently and tragically flawed mantra "the goal of an enterprise is to maximize shareholder value" is starting to be replaced with healthier alternatives (see e.g. Martin: The Age of Customer Capitalism. Harvard Business Review Jan-Feb 2010), it will probably take a while before these pervade the mainstream; so, stay awake here!

[13] Every now and then we interview someone who, when explaining his answer for this statement, starts to wonder whether the personal performance should be rewarded based on a more granular scheme than simply tying possible bonuses to organization-wide goals such as revenue. And no, such a system is likely to be a very bad idea in terms of the performance of the whole (Larman & Vodde 2010).

Compared to the day-to-day project priorities, functional responsibilities may be viewed as an extra workload (Dooley, Lupton & O'Sullivan 2005) – or vice versa. Functional budgets may be misaligned with project resource assignments, or there is a misalignment between project skill requirements and departmental resources (McGrath 1996). Lack of adequate cross-functional working is also common (De Reyck et al. 2005), and there may be confusion and conflict over roles and responsibilities between functional- and project managers (Dooley, Lupton & O'Sullivan 2005).

4.2.7 Health of individual activities' practices

While insufficient for ensuring portfolio management efficiency, the health of management practices on the level of individual activities that take up time from the development people remains crucial (Martinsuo & Lehtonen 2007). This is measured with the following statement:

> Each of our different activity types (e.g. product development projects, customer-specific development, maintenance, deliveries, etc.) has its own practices that work

Note that despite the importance of practices on the level of individual activities, you should not wait to perfect them before setting up explicit portfolio management. On the contrary, establishing the rudiments of portfolio management is actually a prerequisite for effective single-project management (Vähäniitty, Rautiainen & Lassenius 2010). Even if you have the best of project management practices, trashing between projects and other tasks (caused to a large extent by inadequate or missing portfolio management) can render them useless.

4.3 Lifestyle

The Health Barometer measures *Lifestyle* in terms of two categories: Development portfolio structure (Section 4.3.1) and Development portfolio management process and practices (Section 4.3.2); see Figure 4.4 below.

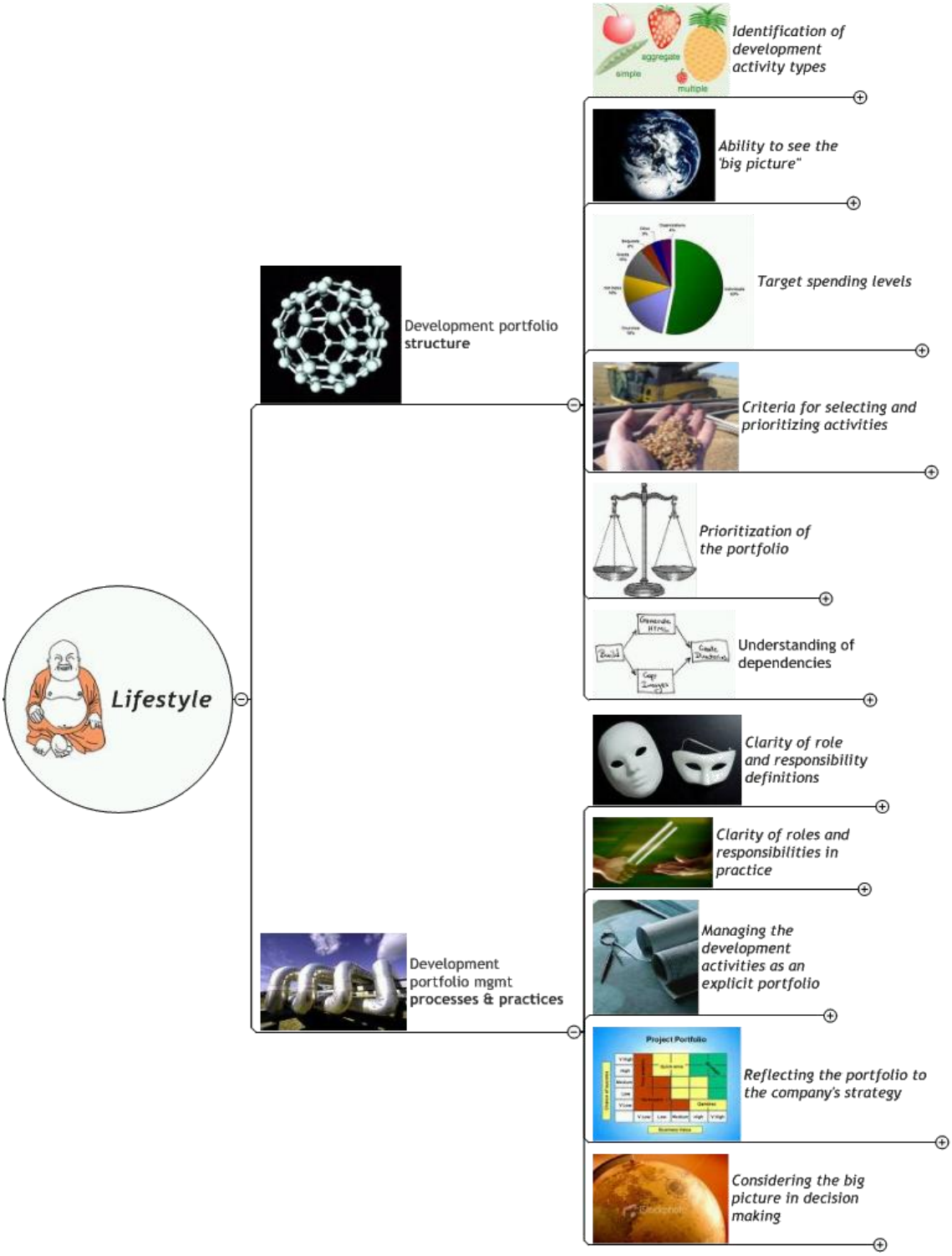


Figure 4.4: Lifestyle as measured by the Health Barometer

A more detailed description on how to improve your lifestyle follows in Part III of this book.

4.3.1 Development portfolio structure

By development portfolio structure we refer to the basic elements used in governing the multitude of potential, upcoming, on-hold, and ongoing development activities. Without a portfolio structure, it is almost certain that one will

end up with a fire-fighting mentality (see Section 4.4.2 for details) as far as prioritization is concerned.

Portfolio structure consists of identifying the types of development activities (Section 8.2.3), setting and monitoring target spending levels (Section 8.2.4), setting the relative priorities of the ongoing activities (Section 8.2.7), setting the criteria for selecting and prioritizing development activities (Section 8.2.7), and understanding the possible dependencies between the activities, and finally visualizing the 'big picture' of ongoing activities (Section 8.2.2). Table 4.1 presents the statements used to measure the health of these issues.

Table 4.1: Statements used to measure development portfolio structure

Issue	Statement(s)
Identification of development activity types	We have identified the different types of activities development people spend their time on (e.g. product development projects, customer-specific development, maintenance, deliveries, etc.)
Ability to see the 'big picture''	Business people are able to see the 'big picture' of ongoing activities (a.k.a. the development portfolio)
	Development people are able to see the 'big picture' of ongoing activities (a.k.a. the development portfolio)
Target spending levels	I understand how much time, from a business perspective, I should spend on different types of activities
Criteria for selecting and prioritizing activities	We have criteria for prioritizing our ongoing development activities
Prioritization of the portfolio	I understand the priorities between ongoing activities (e.g. project X vs. project Y, project X vs. support request Z, etc.)
Understanding of dependencies	I understand the dependencies of the ongoing activities

4.3.2 Development portfolio management process and practices

With respect to the process & practices for development portfolio management, the issues and the corresponding statements used to measure them are displayed in Table 4.2.

Table 4.2: Statements used to measure the process and practices for development portfolio management

Issue	Statement(s)
Clarity of role and responsibility definitions	We have defined who are responsible for development portfolio management
Clarity of roles and responsibilities in practice	It is clear who should, in different situations, participate in development-related decision making (e.g. in the middle of a project, when an urgent maintenance request arrives, when making an offer, when deploying a product, etc.)
Managing the development activities as an explicit portfolio	All the ongoing and immediately upcoming activities that require attention from the developers are managed as an explicit portfolio
Reflecting the portfolio to the company's strategy	We actively reflect the content of the development portfolio to the strategy of the company
Considering the big picture in decision making	In decision making we mainly consider individual activities and do not take the "big picture" into account

How to do agile-compatible portfolio management is explained in Part III. Now, let's look at the symptoms that are associated with inadequate portfolio management.

4.4 Symptoms

In order to get an overview of the symptoms associated with inadequate portfolio management, we conducted a systematic literature review (Vähäniitty, Rautiainen & Lassenius 2010) and distilled the findings into eight problem areas that are symptomatic of inadequate portfolio management (see Figure 4.5 below).

The symptoms are often the most tangible and pressing issues. They can usually be easily observed in a company. They are: 1) excessive multitasking; 2) firefighting; 3) overload; 4) ineffective decision making, 5) missing strategic alignment; 6) slipping schedules; 7) negative changes in performance; and 8) perceived need to improve project management. The symptoms and the statement groups used to assess the degree of their presence are further described in Sections 4.4.1-4.4.8 below.

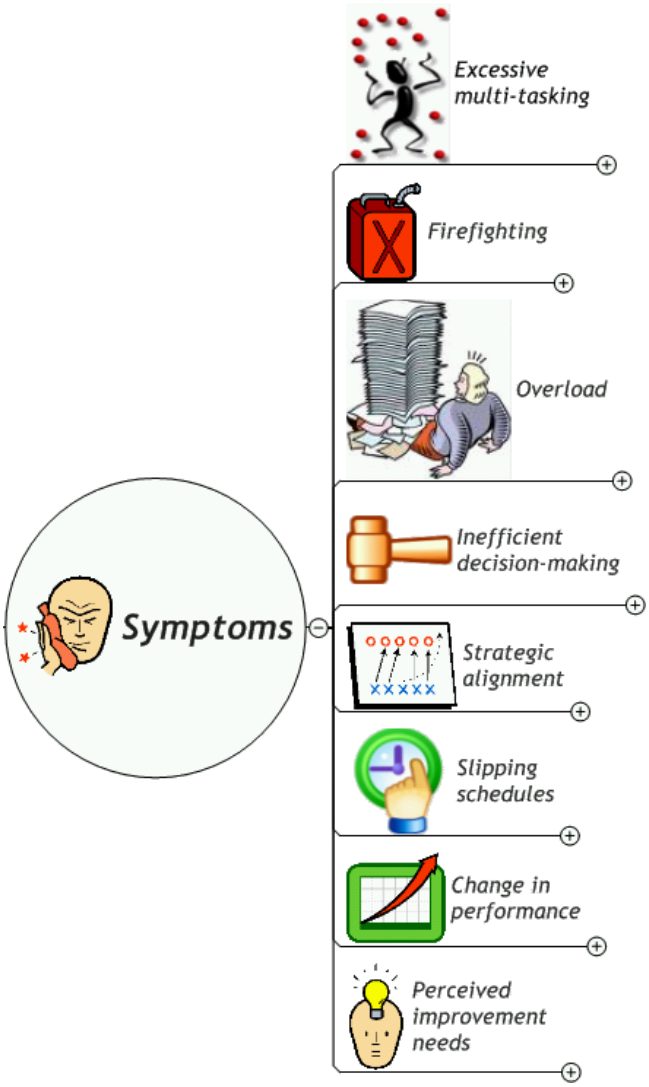


Figure 4.5: Symptoms of inadequate portfolio management as measured by the Health Barometer

4.4.1 Excessive multitasking

Assigning the same individual to multiple parallel projects enables organizations to use the person's expertise for more than one project (McDonough & Spital 2003) and reduces the time that resources are idle (Zika-Viktorsson, Sundström & Engwall 2006, Laslo & Goldberg 2008). The best developers may find themselves assigned to more than four or five concurrent projects (Wheelwright & Clark 1995, Wheelwright & Clark 1992) or crisis management duties (Wheelwright & Clark 1992). People assigned to too many concurrent projects start working in a time-sharing manner in an attempt to show progress on all projects which they are working on (Anavi-Isakow & Golany 2003). The completion of each project is slowed down (Cerveny & Galup 2002), and time is lost due to learning, forgetting, and re-learning (Ash & Smith-Daniels 2004, DeMarco 2001). More time is needed for activities with low value such as update meetings and problem-solving meetings (Cerveny & Galup 2002, Wheelwright & Clark 1995, Kaulio 2008). Excessive multitasking has also been reported to re-

sult in perceiving work as disrupted and fragmented, with less opportunities for recuperation, competence development, or improvement of work routines (Zika-Viktorsson, Sundström & Engwall 2006).

The issues and the corresponding statements used to measure excessive multitasking are listed in Table 4.3.

Table 4.3: Statements used to measure excessive multitasking

Issue	Statement(s)
Number of ongoing activities	How many different activities (product development projects, customer projects, etc.) are currently ongoing in your company?
Number of your own responsibilities	In addition to my main responsibility, I also have other, time-demanding responsibilities
Compromised throughput due to optimized resource utilization	We have too many parallel ongoing activities
Amount of parallel work in general	A single person is usually assigned to only one activity (e.g. a project) at the same time
Intentional limiting of work-in-progress	We complete one thing at a time and don't shift our attention from one incomplete task to another

4.4.2 Firefighting

Firefighting refers to the reactive and unplanned allocation of resources to solve and fix problems that are discovered late in a project or during maintenance. Firefighting is a self-reinforcing phenomenon and sometimes becomes the de facto process for resource allocation: activities must be claimed to be urgent if they are to be attended to at all (Repenning 2001). While the management personnel should have the flexibility to reallocate resources (Cooper, Edgett & Kleinschmidt 1997, Blichfeldt & Eskerod 2008), reactive resource redistribution tends to produce unanticipated negative effects on other projects in the portfolio (Engwall & Jerbrant 2003). The issues and the corresponding statements used to measure firefighting are listed in Table 4.4.

Table 4.4: Statements used to measure firefighting

Issue	Statement(s)
Resource allocation by fire fighting	“Fire fighting” describes our work in practice
Cascading effect of resourcing changes	Changes in resourcing for one activity (e.g. a project) cause uncontrolled changes in other activities
Ignoring resource allocations	Resources are being shifted from one activity (e.g. a project) to another regardless of previously agreed assignments
Flexibility of re-sourcing	Resource commitments are too rigid for leveraging suddenly emerging opportunities

4.4.3 Overload

Resource demands are commonly met by having people work overtime because of the effectiveness of this approach in the short term (Payne 1995, DeMarco 2001). However, often too few people are simply trying to accomplish too much (Cooper, Edgett & Kleinschmidt 2000, Blichfeldt & Eskerod 2008)(De Reyck et al. 2005, Englund & Graham 2001). A typical overload may be two to three times the actual capacity of the workers (Wheelwright & Clark 1995, Wheelwright & Clark 1992). Overload may also occur when a significant amount (up to 50%) of development resource effort is spent on tasks that the developers are not supposed to attend to or that are perceived to have a marginal impact in terms of resource expenditure (Wheelwright & Clark 1992, Blichfeldt & Eskerod 2008). The issues and the corresponding statements used to measure overload are listed in Table 4.5.

Table 4.5: Statements used to measure overload

Issue	Statement(s)
Working overtime	I work overtime
Pipeline management by push	When selling or making offers we do not consider how to resource the work in practice
Launch frequency of new activities	New activities (e.g. projects) are launched too often
Impact of overload to work quality	Our employees have too much to do and quality of work suffers from it
Sufficiency of resources	We have enough resources in proportion to the amount of work

4.4.4 Ineffective decision-making

The term ineffective is used here as an umbrella term for several issues: 1) late, 2) toothless (e.g., lacking clout), and 3) misguided and/or uninformed portfolio-level decision making.

First, the senior management may lack the time or the commitment to participate in portfolio decision making (De Reyck et al. 2005), provide the necessary guidelines (McGrath 1996), or give feedback to guide the projects in the right direction (Wheelwright & Clark 1995). Thus, they deal with problems at the last moment only, if at all (Wheelwright & Clark 1995). As a result, development decisions with strategic implications have to be made by the frustrated developers (McGrath 1996).

Second, ongoing projects may be very hard to terminate (Cooper & Edgett 2003). Projects are seldom stopped (Elonen & Artto 2003), and when they are, they may be put in a "holding tank", an endless list of projects recognized as inferior but which nobody wants to terminate (Cooper & Edgett 2003, Payne 1995, Cooper, Edgett & Kleinschmidt 1997, Mader 2004, Cooper, Edgett & Kleinschmidt 2001). The incentives of the managers or sales people may also be tied to the projects in a dysfunctional way (Cooper & Edgett 2003, Payne 1995).

Third, a common situation is that no relevant data on which portfolio decisions could be based have been collected (Cooper, Edgett & Kleinschmidt 2001). Management may also be overwhelmed with all the possible ways to plot and visualize relevant information (Cooper, Edgett & Kleinschmidt 1997), and the information models used for portfolio-level decision making may imply a degree of precision far beyond the reliability of the actual data (Levine 2005). The issues and the corresponding statements used to measure ineffective decision making are listed in Table 4.6.

Table 4.6: Statements used to measure ineffective decision making

Issue	Statement(s)
Pruning of non-essential activities	Activities (e.g. projects) are never killed
Management involvement in decisions regarding activities	If time runs out, developers resolve by themselves what can be left undone
Monitoring progress of activities	The real status of activities is known in development portfolio -level decision making
Rate of change of priorities	The priority ranking of activities changes constantly
Management response to problems	Management reacts to problems detected in activities (e.g. projects) too late
Senior mgmt's involvement in portfolio level decision-making	Senior management is actively involved in portfolio-level decision making
Health of the dialog between Business and Development	The dialogue between Business and Development people works

4.4.5 Strategic alignment

The ongoing mix of projects may not be strategically aligned (McGrath 1996) or it may lack an apparent link to strategy or organizational goals (De Reyck et al. 2005, Englund & Graham 2001, Cooper, Edgett & Kleinschmidt 2001, Cooper, Edgett & Kleinschmidt 2001). As there is no possibility to make firefighting or project selection decisions in the context of strategy, divergence between individual projects and the goals of the entire organization easily develops (Dooley, Lupton & O'Sullivan 2005, McGrath 1996, Wheelwright & Clark 1992, Cooper, Edgett & Kleinschmidt 2001). A portfolio consisting of many relatively small projects of low value, such as small adjustments and modifications to existing systems, has also been reported to be a sign of missing strategic alignment (Cooper, Edgett & Kleinschmidt 2000, Cooper, Edgett & Kleinschmidt 2001). The issues and the corresponding statements used to measure strategic alignment are listed in Table 4.7.

Table 4.7: Statements used to measure strategic alignment

Issue	Statement(s)
Strategic alignment of ongoing activities	Ongoing activities are in alignment with the company's strategy
Significance of ongoing activities	Ongoing activities are essential to our business
Portfolio balance: leveraging existing products	We have a sufficient amount of development projects that incrementally improve existing products or services
Portfolio balance: creating new business	We have a sufficient amount of product or service development projects that aim for new business

4.4.6 Slipping schedules

Sometimes, projects are late (De Reyck et al. 2005, Wheelwright & Clark 1992, Blichfeldt & Eskerod 2008), time to market is increased (Cooper, Edgett & Kleinschmidt 2001, Cooper, Edgett & Kleinschmidt 2001), and development cycle times are increased (Cooper, Edgett & Kleinschmidt 2000) because of inadequate portfolio management. Target dates do not become commitments, because company workers know that the priorities will shift and the dates will be revised again (Wheelwright & Clark 1995). The issues and the corresponding statements used to measure slipping schedules are listed in Table 4.8.

Table 4.8: Statements used to measure slipping schedules

Issue	Statement(s)
Progress of activities	Ongoing activities are behind schedule
Activity progress status reporting	Progress of ongoing activities is reported optimistically

4.4.7 Change in performance

Negative changes in performance, visible from, for example, project failures and disappointing project outcomes, are often associated with inadequate portfolio management (Cerveny & Galup 2002, De Reyck et al. 2005). Performance may suffer due to compromised project scope and quality, too many low- or high-risk projects, or insufficient penetration of the market. Product launches may be issued in an indifferent manner, and the overall failure rate of products and/or features is high (Cooper, Edgett & Kleinschmidt 2001, Cooper, Edgett & Kleinschmidt 2001). The issues and the corresponding statements used to measure changes in performance are listed in Table 4.9.

Table 4.9: Statements used to measure failures and poor profitability

Issue	Statement(s)
Performance of the development organization	From a business viewpoint, development performs its duties well
Improvement in software development capability	Our capability to produce high-quality software has improved during the past year

4.4.8 Perceived improvement needs

Inadequate portfolio management may not be recognized as a 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 (Cerveny & Galup 2002). While efficient management of individual projects has been found to be important for efficient portfolio management, it is not sufficient to guarantee such efficiency (Martinsuo & Lehtonen 2007). The issues and the corresponding statements used to measure perceived improvement needs are listed in Table 4.10.

Table 4.10: Statements used to measure perceived improvement needs

Issue	Statement(s)
Investing in individual activities' practices	We should invest more in improving the practices of individual activities (e.g. project mgmt., team practices, deployment processes, sales processes, customer support, etc.)
Investing in development portfolio mgmt practices	We should invest more in improving development portfolio management (e.g. prioritizing activities, linking strategy with daily work, structuring the development portfolio, etc.)

Chapter 5: Performing a Portfolio Management Health Barometer Study

Ville Heikkilä & Kristian Rautiainen

This chapter describes how you can perform a portfolio management health barometer study. The first section describes how to prepare for a health barometer round, the second section describes how to collect data, the third section describes how to analyze the data and the fourth and final section describes how to visualize and present the findings from the health barometer. Instructions are given for both the first round and the subsequent rounds. The first round performed in an organization differs somewhat from the subsequent rounds, as the results from the previous round are used as a reference to find changes in the health levels.

5.1 Preparing for a Health Barometer study round

There are a few things you need to do to prepare for a Health Barometer (HB) study round. Depending on whether it is the first round at your organization or a consequent follow-up round, different details need taking care of. In general, you need to select participants, prepare and open the survey, book interview times, and prepare and send instructions. These are discussed in the following sub sections. The terminology we use here refers to the HB tool which was used in the ATMAN project, the details of which are discussed in Chapter 6. Naturally you can use other survey tools available to you, but then you also need to map the terminology to that context.

5.1.1 Selecting the participants

Before you do anything else, you should consider who should participate in a HB round from your organization. The basic idea should be that more people participate in the survey than in the interviews. Since each organization is unique in this regard, it is not possible to give any concrete number of interviews which would suffice to uncover the root causes. However, we have found that six to ten interviews per round usually suffice to uncover causes to the most problematic areas, depending on the size of the organization. You should inter-

view members from different parts of your organization, from different organizational levels, and people with different kinds of work type (developers, managers, testers, etc.) and at least two people with a similar work type and organizational position to identify any outliers[14]. If your organization works on many projects, one idea for sampling participants is to include people that are involved in the same project(s). In that way the participants are more likely to refer to the same context when choosing their answers, providing less "noise" in the answers.

For subsequent rounds, we recommend that you include as many people from the previous round as possible, especially if you choose the option in the Health Barometer survey software to show the participant's answers from the previous round. We have found that this produces results that are more comparable than results when you include a lot of new people in consecutive rounds.

5.1.2 Preparing and opening the survey

If it is the first survey you are conducting, you need to create the questionnaire. The steps for this are:

1. Create the Organization you are studying in the HB tool.
2. Create a new Round for your Organization by choosing the ATMAN Default Round and cloning it to your Organization. In this way your starting point is the set of Sections, Issues, and Statements which were created and used in the ATMAN project. These are explained in Chapter 4 and Chapter 6. Leave the Round in Draft state.
3. Check the list of Responsibilities and groom it to match your Organization. If some Responsibilities are missing from the list, you can create new ones.
4. If you feel like adding new Statements, go ahead. However, be careful about creating too long a questionnaire.
5. Tweak the questionnaire to your liking or just go ahead and use the ATMAN Default. When you are ready, change the State to Started to open the questionnaire for the participants. Remember to Save.
6. If you are using some other tool, check the ATMAN Default questionnaire in English and Finnish in Appendix B and Appendix C.

On subsequent rounds you do not need to create your organization and you should use your previous round as basis for cloning. In this way any tweaks you might have made to the questionnaire are included. You can also choose whether you want to show the answers of the previous round to the participants when they are filling in the questionnaire. This is done by selecting the reference

[14] Outlier is a data point that is greatly different from the rest of the results. For example, an employee that has received a notice of termination might not represent his/her demographic group accurately. Naturally, you should avoid interviewing such employees.

round to be the previous round number or “(none)” for not showing previous answers. Otherwise, the steps described above apply also to subsequent rounds.

5.1.3 Booking interview times and the dissemination time

Now you have opened your questionnaire. Before you send instructions to the participants, you need to book the interview times and the dissemination time. Of these the interview times are more important, because you need to ensure that the selected interviewees are available. The dissemination time can always be agreed later on.

You should reserve at least one and half an hour for the interviews. For the first round you may even consider reserving two hours. On subsequent HB rounds the issues are more familiar which makes for a speedier interview. For the person(s) responsible for performing the interviews it may be smart to stack as many interviews on the same and/or consecutive days as possible as the analysis can then be started with all the data fresh in mind.

The dissemination time should be chosen so that as many people can attend as possible. Discussing the results and improvement suggestions gives a chance to find more results and improvement suggestions and refine the existing ones, creating an atmosphere of shared ownership. In that way the dissemination event is also an important part of the analysis of the results and the whole portfolio management improvement effort.

5.1.4 Preparing and sending instructions

While answering a questionnaire in itself is no rocket science and thus would not require instructions, four important areas need addressing in the instructions: confidentiality, terminology, deadlines, and where to go to find the survey questionnaire. If you want honest answers, you need to provide sufficient confidentiality to the participants. In some organizations this may not even be an issue, but at least give it a thought. In the ATMAN project the context for confidentiality was straightforward. The researchers performed the interviews and analyses and only reported aggregated findings without disclosing answers from any single participant to the companies. For a self assessment of an organization, there needs to be a trusted person in charge of the data, so that the participants can feel safe in giving their answers.

At least for the first HB round the terminology may not be familiar and thus needs to be explained in the instructions. For the consequent rounds, there may be some changes in how your organization interprets the terminology, and those should naturally be explained.

Deadlines for answering the questionnaire should be set. Otherwise you may get very few answers. For the interviewees the deadline should be before the agreed interview time, so that there are answers to walk thorough in the interview.

However, we have also conducted interviews where the interviewee had not filled in the questionnaire beforehand. The interviewee "answered aloud" during the interview, which also seemed to work well.

The final important detail in the instructions is a pointer to where the questionnaire can be found. You should also include some motivation in the instructions. In Appendix A we provide an example of the instructions used in the ATMAN project that you can use as a starting point for your own instructions. When the instructions are ready, you should send them to all participants. Depending on the level of confidentiality you need, consider using the bcc field in your e-mail software.

5.2 Gathering Health Barometer data

5.2.1 Health Barometer questionnaire

The purpose of the Health Barometer questionnaire is to gather quantitative information from the people in your organization about the current state of development portfolio management. You should try to get as many people as possible to answer to your questionnaire, since people with different backgrounds can produce interesting results. You might feel that, for example, technical writers do not need to participate, but we still recommend that you include them at least during the first Health Barometer round. During the second and following rounds you may want to narrow your selection of participants to those who provided the most interesting results during the previous rounds, although there is little harm[15] in inviting your whole organization to answer to the questionnaire.

5.2.2 Health Barometer interviews

The goal of the health barometer interviews is to uncover the root causes of the most problematic issues in the organization. Every interviewee should have completed the Health Barometer questionnaire before being interviewed. Before starting the interview have the questionnaire answers from the interviewee open and visible for you and the interviewee. Health barometer interviews are performed in a semi-structured fashion in which the questionnaire provides the interview structure. You start with asking the interviewee to explain what his/her responsibilities in the organization are. You then ask the interviewee to explain his/her answer to each questionnaire statement one-by-one. During the second and later rounds you can also ask why the interviewee has changed his/hers answer to the statement (if it has changed). You should either record the interview for later transcription or write down the answers immediately. For the sake of analysis of the data you should also record the issue number the in-

[15] Filling the Health Barometer questionnaire naturally requires some effort from the respondents. We have found that 30 minutes is typically sufficient for filling the questionnaire. In addition, analyzing the questionnaire can take more time when you have more answers.

terviewee is answering to so you can easily connect the interview answers to the numerical data you get from the questionnaire. If time allows, feel free to ask follow-up questions to questionnaire statement questions when you think the interviewee might produce additional insights if probed further.

One way to probe further into a question during a semi-structured interview is called laddering. You perform laddering by asking follow-up questions using the interrogative words "why" and "how". Why-questions allow you to climb up to more abstract answers while how-questions allow you to climb down to more concrete answers. A helpful example follows:
A: "After each day I record my working hours before I leave the office."

Q: "*Why* do you record your working hours?"
A: "I don't really know, but we are told to record our hours each day."

Q: "*How* do you record your working hours?"
A: "I open our timesheets program and enter my working hours there"

During the first Health Barometer round in your organization you should try to cover every questionnaire statement to build a solid baseline for further rounds. During the second and later rounds you should concentrate on the issues which have changed most, or to those statements that you have tried to affect but which have nevertheless stayed the same or even gone worse. We have found that one and half an hour to two hours per interviewee is usually enough time to cover every statement and also perform some additional probing. If you have less time during the first Health Barometer round you should concentrate on the issues that are most interesting based on the questionnaire analysis results (see Section 5.3). We have conducted interviews lasting 45 minutes, but those gave us no opportunity to investigate any issue deeper and some issues needed to be skipped completely, leaving us with many unanswered questions in the analysis. Therefore we recommend that you try to stick to reserving at least one and half an hour for each interview.

It is always useful to ask the interviewee to provide concrete examples of what (s)he means. In this way you can avoid misunderstandings between the interviewer and the interviewee.

One useful concept is the "theoretical saturation point", which is the point after which new data doesn't add new significant findings. During the interviews you might get a feeling that the answers you get to an issue do not reveal any new information about the issue. In such case you should skip the issue in further interviews and concentrate on those issues you think you need more information on.

In the ATMAN project we interviewed from 4 to 11 participants when 12 to 32 had answered the survey. However, we found that interviewing 4 respondents was not enough and we would have preferred to have more information regarding many issues.

5.3 Analyzing the health barometer data

5.3.1 Preparing the questionnaire data

It is important to remember that some statements in the health barometer questionnaire are inversed. If your survey software doesn't automatically inverse the results from the inverse statements, you should take care to do it yourself before you start the analysis of the numbers.

This section is written with the expectation that you use Microsoft Excel spreadsheet tool to analyze the questionnaire results. You should keep the results sheet as it is and make duplicates of the sheet for further manipulation of the data. During the second and further rounds you should copy the original answers sheet from the previous round to the same workbook with the current round answers sheet to make the comparison between rounds fluent (see Section 5.3.4). Whenever you want to analyze a subset of the results you should duplicate the original results sheet and conduct the analysis using the duplicate sheet.

The file exported from the Health Barometer questionnaire software is in Microsoft Excel file format. To make the analysis of numerical results of a health barometer round as straightforward as possible the raw exported results need to be prepared for analysis. The exported file contains only one sheet called answers. The sheet is in tabular format where each row contains either one issue or median of several issues. Each column contains the answers of one respondent. You should copy the answers sheet into a new sheet called numbers. The numbers sheet contains several columns which we do not need for the numerical analysis. The "Answer" column contains the numbers we are interested in, so you can delete "Original", "User comment" and "Admin comment" columns for each respondent.

5.3.2 Calculating and analyzing median and dispersion

The HB questionnaire uses Likert-like scale and the results should be analyzed as ordinal level data. Calculate the *median answer* to each statement and the *dispersion of the answers* using interquartile range[16]. Since the numerical answers fall between 1 and 6, the interquartile range is between minimum of 0 and maximum of 5. Typically interquartile range of 2,5 or over means that you

[16] Interquartile range is the difference between the 75% quartile and 25% quartile values.

should look at the issue more closely. Figure 5.1 illustrates the relationship between median and interquartile range. Issues that have low dispersion are the most straightforward to analyze, since most of the respondents agree on the current situation. You should identify 3-5 most problematic (high median) issues with low dispersion and try to find out the root causes for the problems by analyzing the interview responses to those issues (see Section 5.4).

Most statistical software packages have a function for calculating the interquartile range. You can also calculate it using Microsoft Excel function QUARTILE: QUARTILE(results;3) - QUARTILE(results;1) where results is the cells containing the results of a single questionnaire statement.

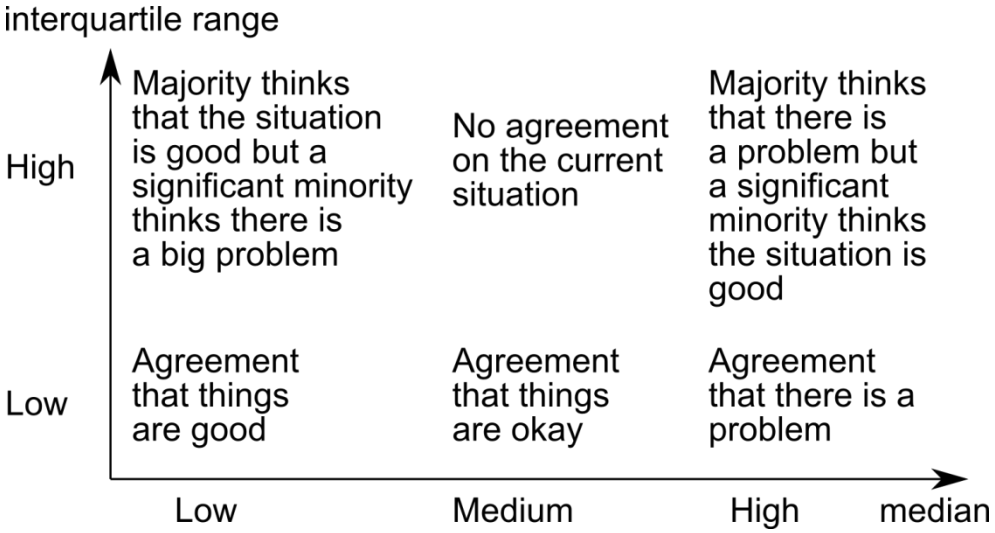


Figure 5.1: The relationship between median and interquartile range

If you find issues with high dispersion you should try to identify commonalities between the respondents who dissented from the majority. Typical commonalities are related to organizational units, position in the organization, or the type of job. For example, you might notice that the answers to the issue “Working overtime” have high dispersion but low median and further analysis could reveal that everyone who agreed to the statement belongs to the same team. You should then look at the interview responses of the team members to find out why they are working overtime (see Section 5.4).

5.3.3 Comparing demographics

In the previous section we tried to find demographic groups by analyzing the data to identify commonalities between respondents. You can also define the demographic groups before you look at the data. Group the respondents that belong to the same demographic group and calculate the median answers and dispersion for each group. By looking at the median and dispersion you can identify issues that have a notably different median between the groups and a low dispersion inside the group. Differences of one point or more in the median

answer of the two respondent groups typically mean that there is notable difference of opinion between the groups. You should note such issues and try to find explanations from the interview results. Some groups you could look at are respondents with different types of jobs, for example developers versus managers, and respondents from different workplace locations. Table 5.1 shows an example of demographic group comparison. There is a 1,5 point difference in the median between managers and developers regarding issue "Appropriateness of organizational structure", but the interquartile range (dispersion) of developers' answers is quite high. Thus you cannot conclude that developers in general think the situation in much worse than the managers. The two-point difference in the median regarding issue "Clarity of roles and responsibilities" together with the low interquartile ranges suggests that there is a clear difference of opinion between the two groups regarding the issue.

Table 5.1: An example of a demographic group comparison

	MANAGERS		DEVELOPERS	
Issue	**Median**	**Interq. Range**	**Median**	**Interq. Range**
Appropriateness of organisational structure	3,5	1	5	2
Clarity of roles and responsibilities in practice	5	0,25	3	1

5.3.4 Comparing Health Barometer rounds

When we compare two rounds of health barometer we are more interested in the changes between the rounds and more specifically changes in the answers of the respondents who participated in both rounds. Naturally such comparison is only possible if you have a sufficient number of such respondents. In addition, you should only include the answer of a single person to a single statement if the person has answered to the statement on both rounds. If a person has not answered to the statement (empty result) or answered "Don't know" (N/A result) on either round you should exclude the person from the median and interquartile range calculations of that statement. Analyzing the changes is straightforward; you simply compare the medians of the issues between the rounds to find any notable changes and note those issues for the interview analysis (see Section 5.4). If you received enough answers to the health barometer questionnaire you can also combine the demographic analysis and round comparison to identify demographic groups that have changed their opinions between the rounds. Such groups have great potential for interesting findings in the interview analysis.

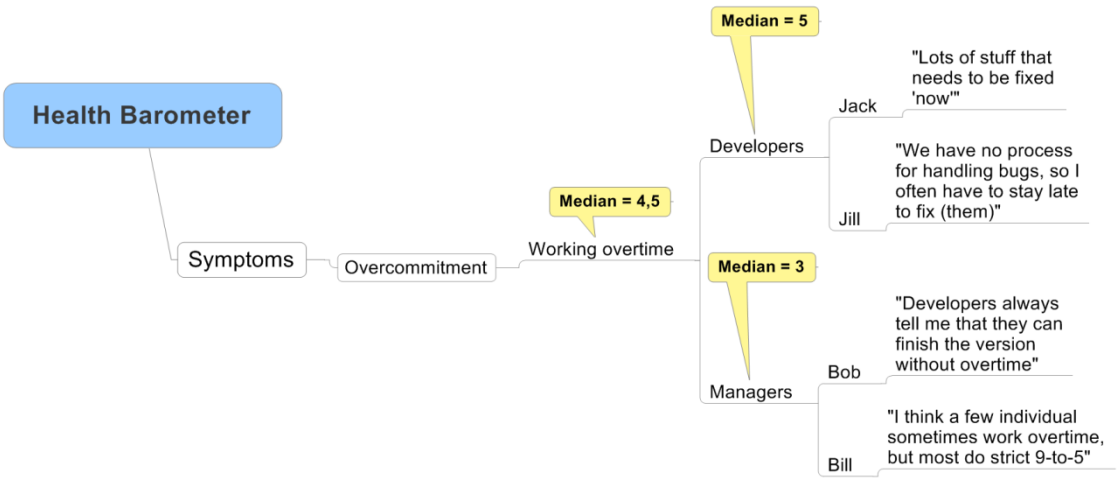
5.4 Analyzing the interviews

Interview analysis relies heavily on the intuition and insight of the analyzer. The basic process is simple. You first arrange the transcribed interviews so that the responses to each statement can be easily compared between the respondents.

We have found that having a large spreadsheet that contains all transcribed interviews works well. We arrange the spreadsheet so that each column contains the answers from a single respondent and each row contains the transcribed answer to one statement.

Next, you should tag the statements that were identified as somehow interesting in the questionnaire results analysis (see Section 5.3). For example, you should try to find out what are the root causes for the most negative statements, why a certain demographic group differs in opinion from the rest of the respondents, or why some issue has improved or degraded notably between two health barometer rounds. Especially you should look for similar answers from several respondents and opposing answers. You should also look at the comments given in the questionnaire for additional insights. Whenever you find interesting or revelatory answers you should note the answer or part of it (i.e. quotation), the reason you think it is interesting or revelatory and, if relevant, the related numerical questionnaire result. The systematic way of processing interviews in such manner is called coding interviews. The final step is to look at your notes and draw conclusions based on the questionnaire results and interview answers. For more instructions on analyzing interviews, see for example Patton (2002).

We have found that compiling the interview notes in a single mind map helps to keep the results organized. The figure below shows an example of how you can record interview notes in a mind map.



5.5 Presenting the results

5.5.1 Interpreting the numbers

When presenting individual numbers you should use textual interpretations of the results in addition to the raw numbers. The scale used in the questionnaire is from “strongly agree” to “strongly disagree”, or in numbers from one to six, respectively. Table 5.2 describes how you can use different textual representations of the different numbers to give your audience a better idea of how the different health barometer result numbers should be understood. The first column contains the numbers which are retrieved from the Health Barometer questionnaire software. The second, third and fourth columns contain textual interpretations of the results for issues from different categories.

Table 5.2: Textual interpretations of the HB numbers

Number	Hereditary issue	Lifestyle issue	Symptom
1	Perfect	Exemplary	Fit as ever
2	Good	Got it covered	Feeling all right
3	Moderate	Reasonable	I’ve been worse
4	So-and-so	So-and-so	Feeling a bit queasy
5	Predisposition for problems	Clearly room for improvement	Ouch, it clearly hurts
6	The only option is to lead a strictly healthy life	High risk and ready to crash	Hospitalized

The answers of individual respondents to the questionnaire are summarized by taking the median of all answers. When there is no single middle value, the median is created by taking the mean of two middle values, which may result in a decimal number. In that case you can either state that the result is between two values, for example 2,5 could be “between moderate and good”, or you can create your own textual interpretation of the value.

5.5.2 Visualizing the numbers

When you present multiple related results together you should try to visualize the numbers using graphic charts. This can help to better identify problematic areas, areas that work well and, in the case of second or later rounds, highlight changes in the results. When you present the results of the first HB round you can use a bar chart to visualize the results. Figure 5.2 gives an example of visualizing Symptoms. The numbers on the vertical axis have been replaced by corresponding textual interpretations of the results from Table 5.2. The values of different issues are shown as the bars on the vertical axis. Note that the smaller a bar in the chart is, the better the current situation regarding that issue is. Looking at the figure, you can easily see that “Strategic alignment” is doing well,

but there are lots of ”Perceived improvement needs” and “Slipping schedules” are a problem.

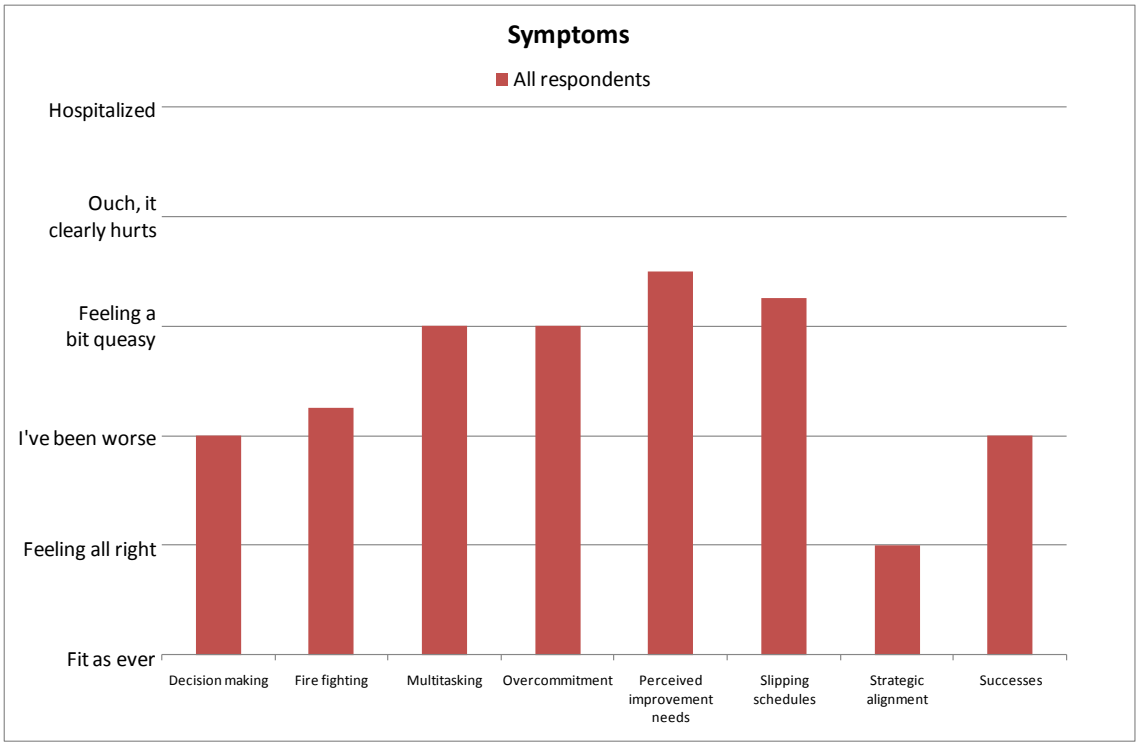


Figure 5.2: Visualizing results with a bar chart

Bar charts are very good for visualizing sets of unrelated data. When you are visualizing multiple series of related data you should use a chart such as the *radar* or *spider web* chart which help you visualize the overall score and compare multiple data series. For example, you can use a radar chart to visualize the medians and dispersions of the component issues of one symptom. Figure 5.3 shows an example of a radar chart visualization of the Overcommitment symptom results. Each spoke of the radar shows the results for one issue related to Overcommitment and the further away from the center the data point is the worse off the issue is. The red line is the median answer, the blue line is the 25% quartile and the green line is the 75% quartile. The top spoke shows the textual interpretations of the result numbers. From the figure you can easily see that “Impact of busyness to work quality” and “Pipeline pushing” are slightly problematic areas according to most respondents, “Working overtime” and “Launch frequency of new activities” are not a problem, and that “Sufficiency of resources” is neutral on average but the answers have a quite high dispersion.

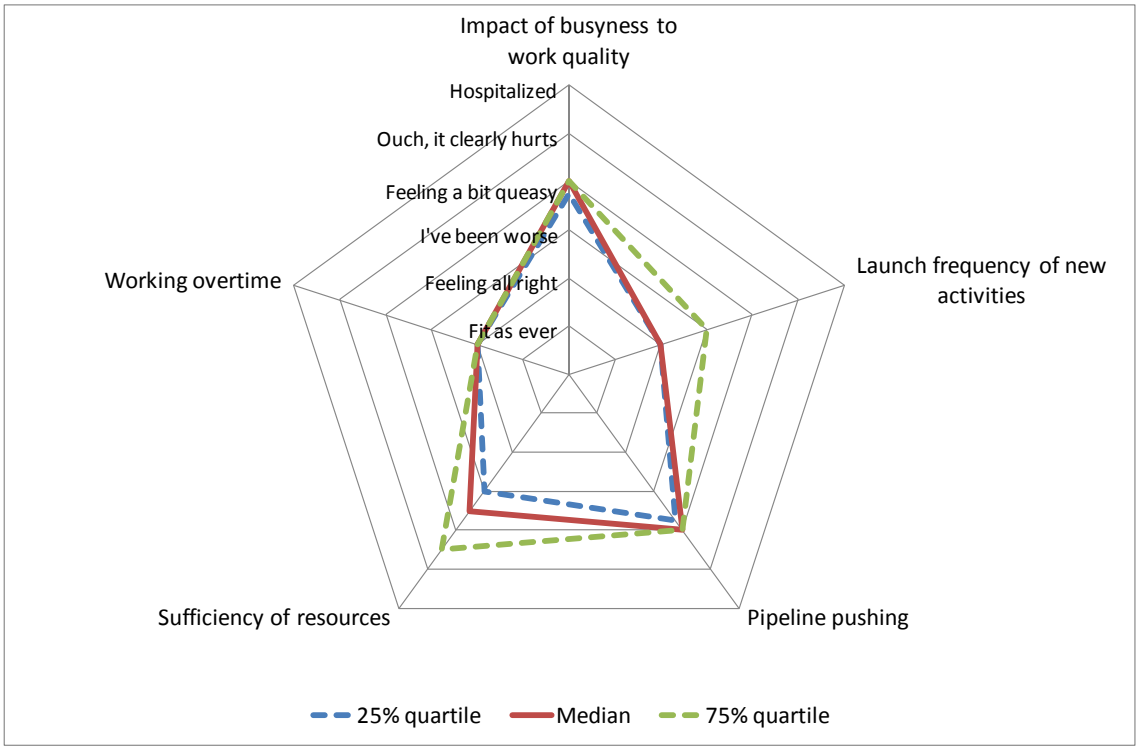


Figure 5.3: An example of Overcommitment radar chart visualization

Radar charts are also excellent for visualizing differences between different demographic groups or visualizing changes between rounds. Figure 5.4 shows an example of visualizing demographic data. You can easily see from the figure that testers are most worried about "Impact of busyness to work quality", developers are worried about "Sufficiency of resources" and in general managers are least worried of the three demographic groups. Figure 5.5 shows an example of visualizing changes over two rounds. Looking at the figure, it is easy to see that "Impact of busyness to work quality" and "Working overtime" have decreased while "Sufficiency of resources" has become worse.

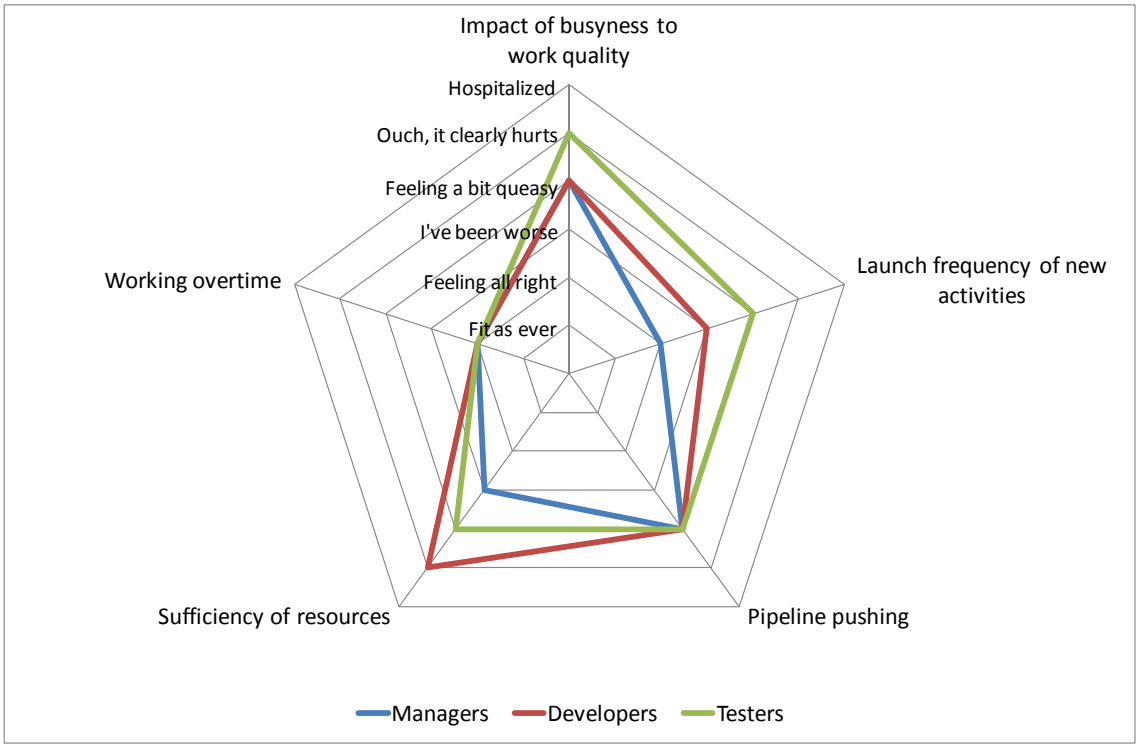


Figure 5.4: An example of visualizing demographic differences

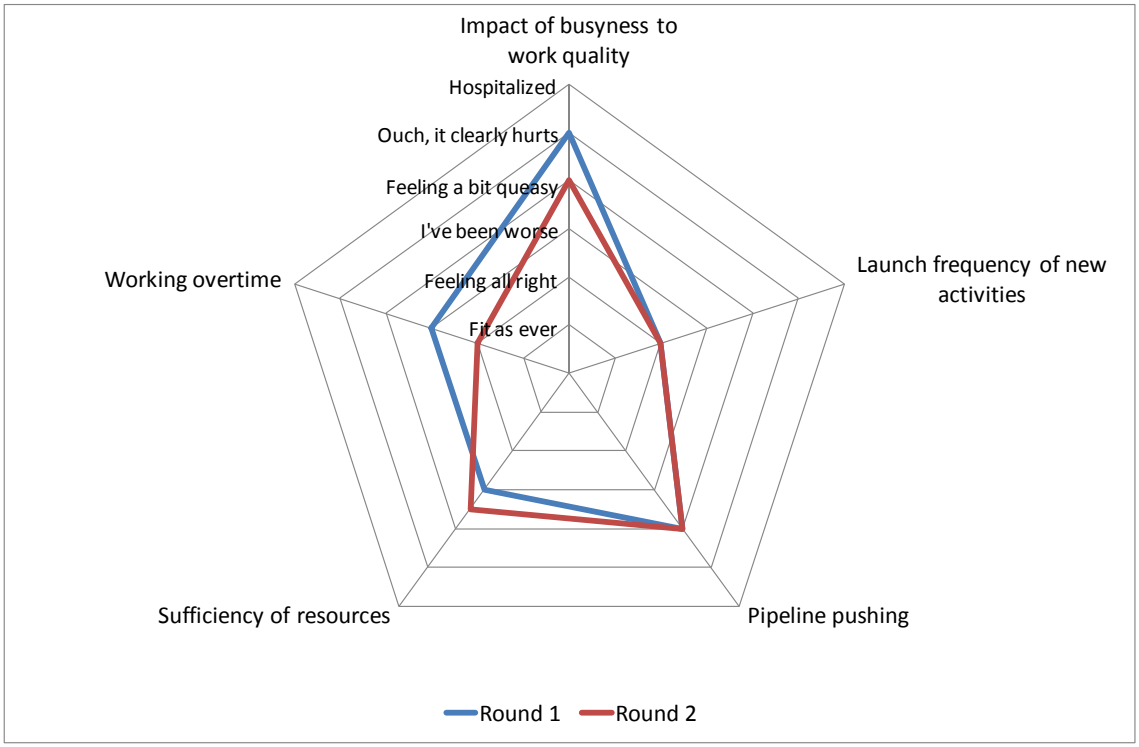


Figure 5.5: An example of visualizing changes over rounds

5.5.3 Presenting the interview results

Regardless of how you have recorded the interview analysis results in your notes you should prepare a separate presentation or report of the results in a more accessible format for your target audience. For example, you can use a structure where you first present an interesting numerical result and then present an ex-

planation of the numbers based on the interviews. If possible, you should give both your own interpretations of the results and direct quotes from the interviews that support you interpretations. Depending on your audience, you also might want to validate your findings and interpretations, for example by asking the members of the audience to raise their hand if they find your results to be believable. You can also ask the audience to help you further analyze the results by asking them what they think is the underlying reasons or root causes of some results.

5.5.4 Then what?

Performing a Health Barometer study is only the first step in a portfolio management improvement process. You might want to discuss the findings immediately after or during your dissemination session or you might defer the discussion to a later time with a different group. A Health Barometer study gives you some ideas of what are the most problematic areas in your organization, but the cures to those areas vary greatly between different organizations. Generally speaking, you should try to affect the lifestyle issues. Changing hereditary factors is usually slow, if possible at all. Curing symptoms is usually useless without first changing the hereditary or lifestyle factors that cause them.

Chapter 6: The Health Barometer Tool

Kristian Rautiainen

This chapter presents the Health Barometer (HB) tool that was created during the ATMAN project for gathering data about the health of organizations' portfolio management. The tool is open source (MIT license) and can be downloaded from the ATMAN Blog. Section 6.1 shows where you can find it. Sections 6.2-6.3 show how to use the HB tool.

6.1 Where to find the HB tool

The installation package can be found from:

atman.agilefant.org

The package contains the installation instructions and the HB tool with the *ATMAN Default* questionnaire included in the database. Additionally you need to install Java 6, MySQL 5, and Tomcat 5.5, but that is all explained in the installation instructions (readme.txt file in the zip-folder).

In the rest of this chapter we explain how to use the tool as an administrator and as a participant in a HB survey.

6.2 Administration tasks

Administration rights are needed for creating surveys and extracting the data from the HB tool. When an administrator logs in, (s)he gets an overview of the existing rounds according to their status (started, draft, closed), as shown in Figure 6.1.

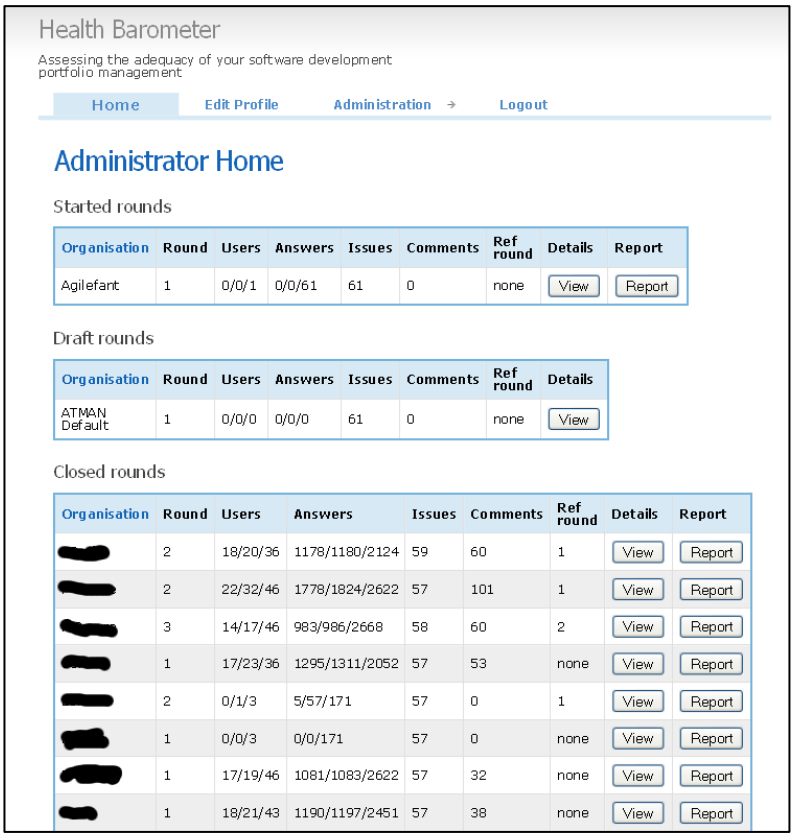


Figure 6.1: Administrator front page in the HB tool

The columns of Figure 6.1 are explained in Table 6.1.

Table 6.1 Explanation of the columns in the administrator front page

Column	Explanation
Organisation	The name of the organization for which a survey has been created.
Round	Round Number. The numbering can be running (e.g. 1, 2, 3, ...) or free.
Users	The number of users that have completed the survey / started the survey / the total number of users who have an account.
Answers	The number of answers that have been completed / the number of answers that should have been completed (# of users who have started the survey multiplied by # of issues) / the maximum number of possible answers (# of users who have an account multiplied by # of issues)
Issues	The number of issues asked in the survey.
Comments	The number of comments in the survey provided by the users.
Ref round	The number of the reference round for the survey.
Details	Clicking on the View button opens a summary of the details for that round (see Figure 6.2), which also shows all the users with an account and the rounds they have attended. From the Round details view you can delete the answers for any user for that round, if needed for some reason.
Report	Clicking on the Report button sends the round's questionnaire form data to a Microsoft Excel file. From a dialog you get to choose if you want to save it on a hard drive or open it in Excel.

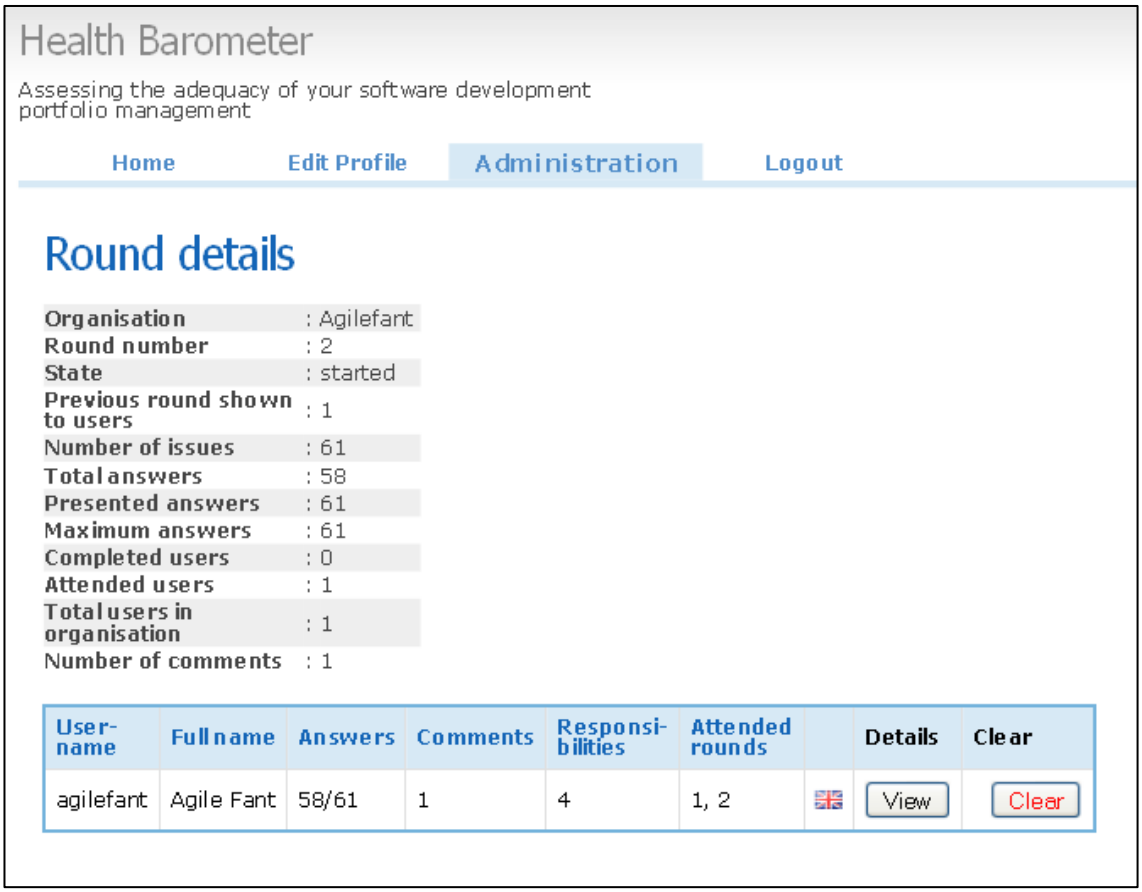


Figure 6.2: Example of the Round details view

From the Administration menu, the administrator can choose different tasks to do. The list of tasks is shown in Figure 6.3.

Figure 6.3: Administrative tasks

The administrative tasks have to do with creating a questionnaire for the HB survey, maintaining the user base, and extracting the data from questionnaire. These are discussed in more detail in the following subsections.

6.2.1 Creating a survey

A survey is organization- and round-specific, which means you first need to create an organization. This is done by selecting Organisations from the Administration menu. This opens a view of all the organizations shown in Figure 6.4.

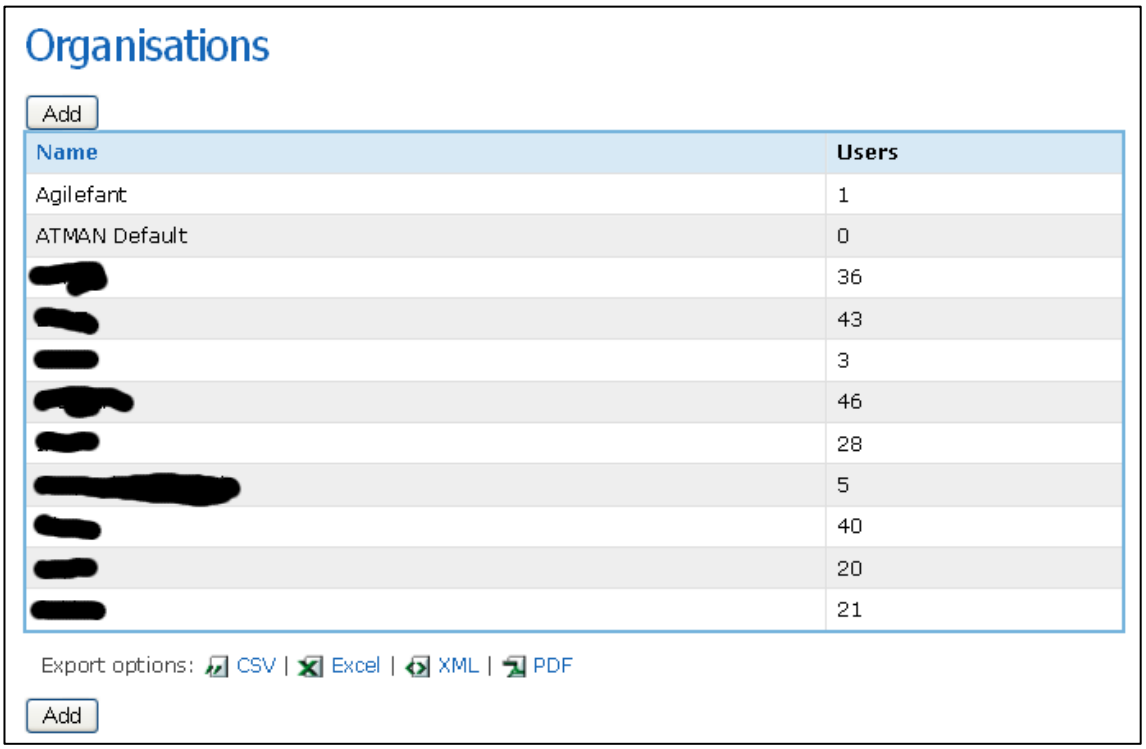


Figure 6.4: Organisations view

A new organization is created by clicking the Add button on the bottom left. This opens a text box where you can write the name of the organization and then save it. Then you need to create a survey round for your organization. This is done from the Rounds view selected from the Administration menu. In the Rounds view (Figure 6.5) you see all the Rounds that have been created in the instance of the HB tool you are using. If you are using the installation package provided by the ATMAN project, you should see the ATMAN Default round in the list. If not, you should first create the Sections, Issues, and Statements for the questionnaire of your survey. These are explained later in this subsection.

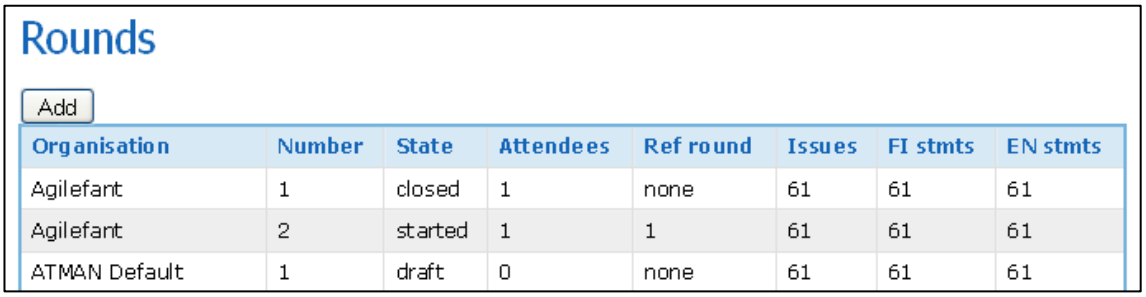


Figure 6.5: Rounds view

If you are creating your first round and see the ATMAN Default round in your list (it should be the only one if you used the ATMAN installation package), click on it to open the Round Information view, which is shown in Figure 6.6. Scroll down the page to find the Clone button. Select your organization from the drop down list left of the Clone button and click Clone. This will open a new Round Information view for your organization (in draft state) with all the ATMAN Default Issues and Responsibilities (see Figure 6.7) already added. If you are happy with that, you can open the round by changing the state to Started. If not, you can edit a few things in the Round Information view:

1. You can move the Issues in different order with the arrow symbols on the right-hand side.
2. You can remove Issues by clicking the red cross furthest to the right.
3. You can choose which Responsibilities you want to exclude to be shown to the users by un-checking the checkbox left of the responsibility in the list (they should all be checked if you cloned the ATMAN Default round). At this stage you might notice that some responsibilities crucial to your organization are missing. These need to be added in the Responsibilities view. Save your round in draft state and go add the necessary responsibilities. You will find your round later from the Round view.

If you need to change any Sections, Issues, or Statements, you need to do it from the corresponding views. Always remember to Save your round and leave it in Draft state, until you are ready to open it.

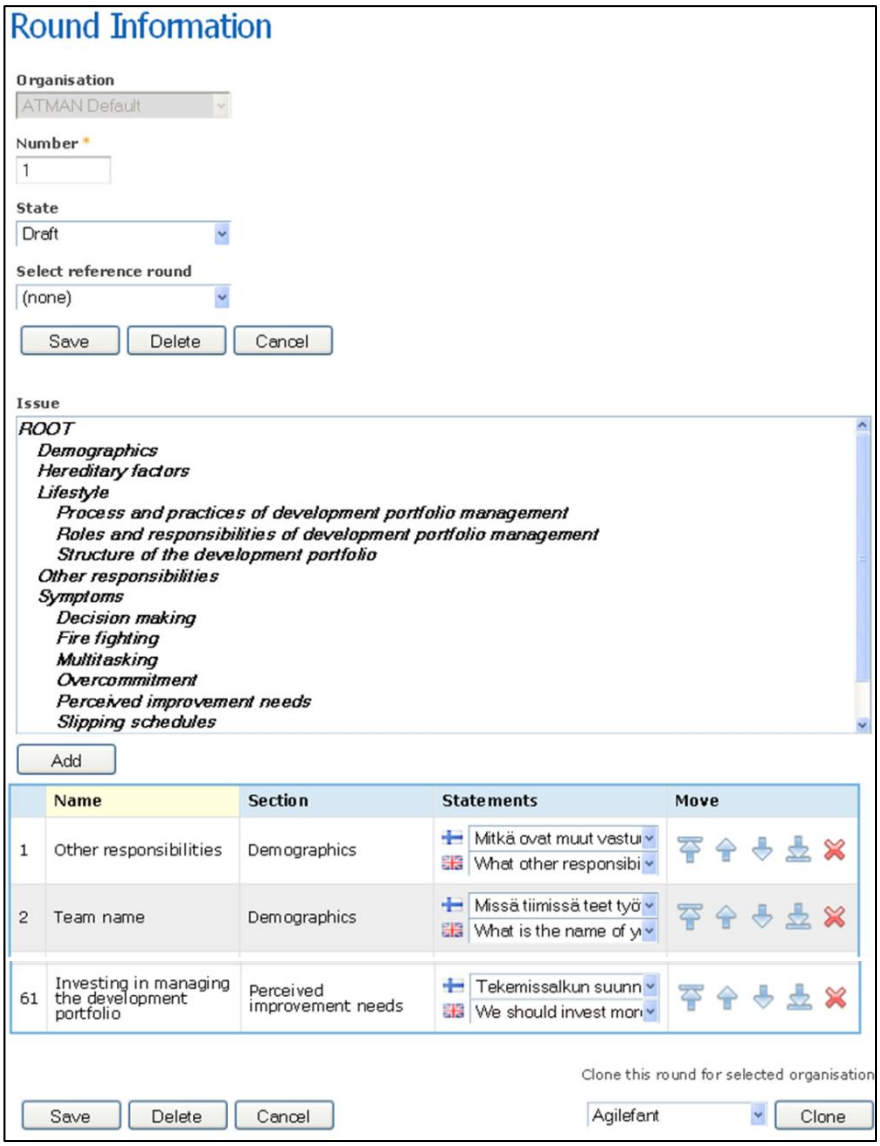


Figure 6.6: Round Information view (truncated in the middle and the end) of ATMAN Default round

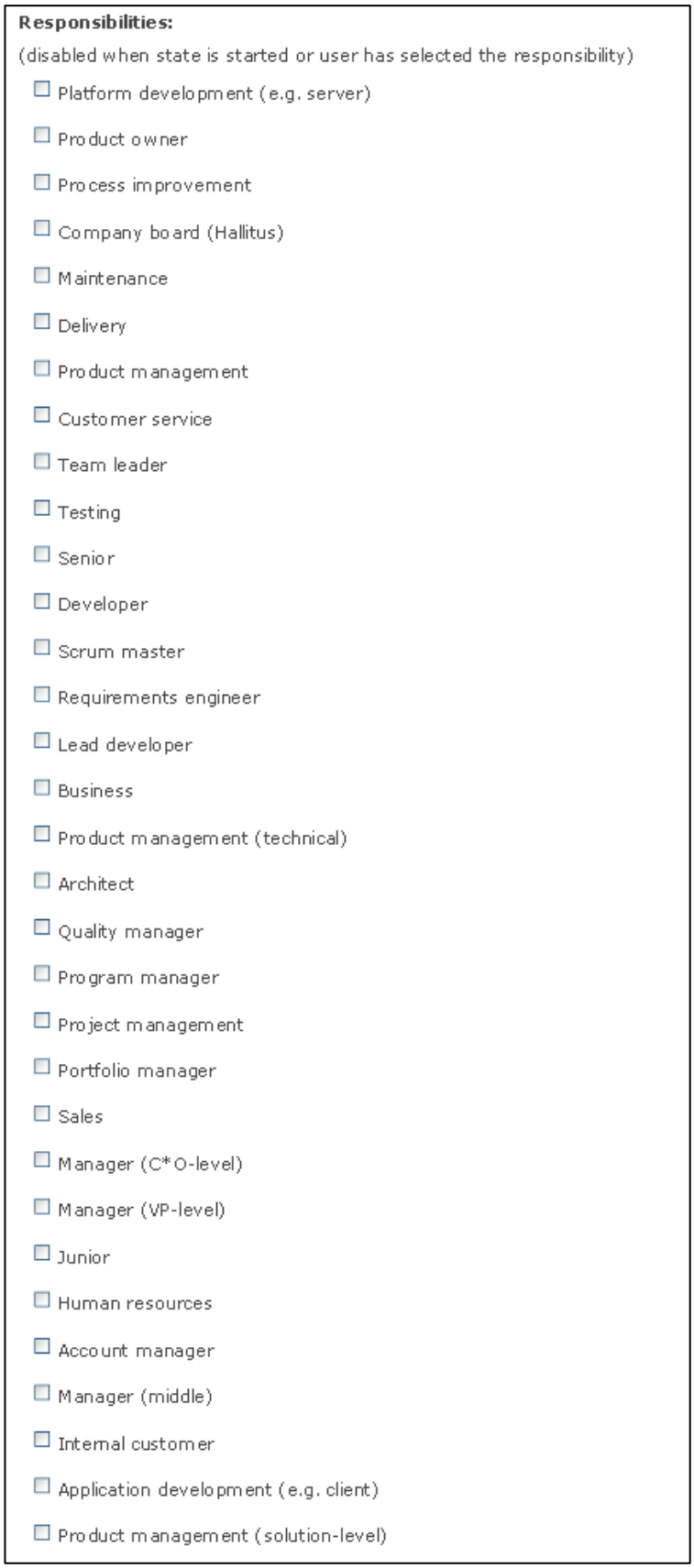
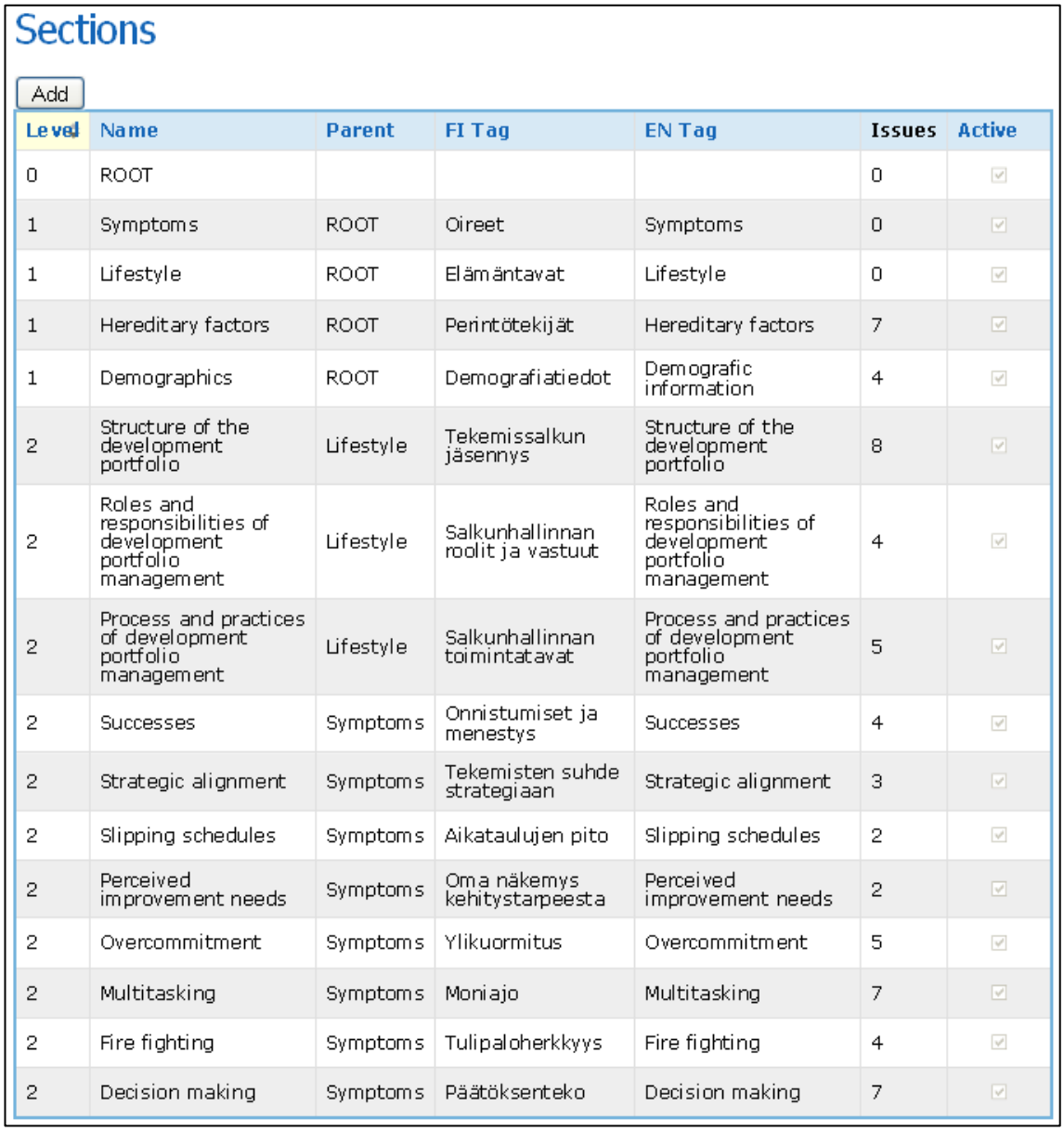


Figure 6.7: Responsibilities list from ATMAN Default

The questionnaire form in a survey is built of Sections, Issues, and Statements. Sections are high-level groupings of Issues. In the ATMAN Default round the

highest level Sections are Demographics, Hereditary factors, Lifestyle, and Symptoms (see Figure 6.8). The three latter are explained in detail in Chapter 4. The Sections can have sub sections, as is the case with both Lifestyle and Symptoms in the ATMAN Default round.



Sections

Add

Level	Name	Parent	FI Tag	EN Tag	Issues	Active
0	ROOT				0	
1	Symptoms	ROOT	Oireet	Symptoms	0	
1	Lifestyle	ROOT	Elämäntavat	Lifestyle	0	
1	Hereditary factors	ROOT	Perintötekijät	Hereditary factors	7	
1	Demographics	ROOT	Demografiatiedot	Demografic information	4	
2	Structure of the development portfolio	Lifestyle	Tekemissalkun jäsennys	Structure of the development portfolio	8	
2	Roles and responsibilities of development portfolio management	Lifestyle	Salkunhallinnan roolit ja vastuut	Roles and responsibilities of development portfolio management	4	
2	Process and practices of development portfolio management	Lifestyle	Salkunhallinnan toimintatavat	Process and practices of development portfolio management	5	
2	Successes	Symptoms	Onnistumiset ja menestys	Successes	4	
2	Strategic alignment	Symptoms	Tekemisten suhde strategiaan	Strategic alignment	3	
2	Slipping schedules	Symptoms	Aikataulujen pito	Slipping schedules	2	
2	Perceived improvement needs	Symptoms	Oma näkemys kehitystarpeesta	Perceived improvement needs	2	
2	Overcommitment	Symptoms	Ylikuormitus	Overcommitment	5	
2	Multitasking	Symptoms	Moniajo	Multitasking	7	
2	Fire fighting	Symptoms	Tulipaloherkkyys	Fire fighting	4	
2	Decision making	Symptoms	Päätöksenteko	Decision making	7	

Figure 6.8: The Sections view

You create Sections in the Section view (from the Administration menu) by clicking the Add button. This opens the Section dialog shown in Figure 6.9. Fill in the name of the Section and the Finnish and English Tags of the Section, which are shown in the questionnaire form to the user. First-level sections have ROOT as parent, sub sections have their corresponding higher-level section as parent, which you choose from the Parent drop down list.

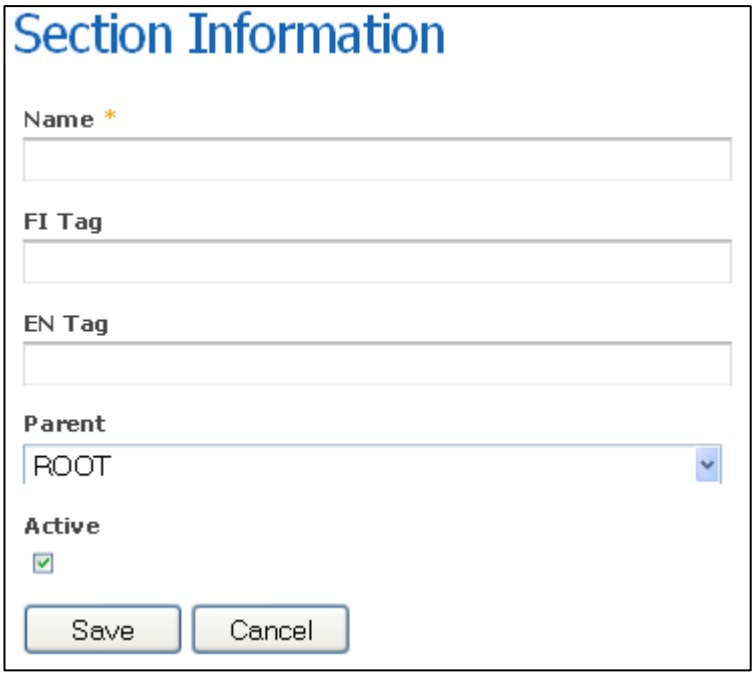


Figure 6.9: Section Information dialog for creating or editing Sections

When you are ready creating the sections you need, you need to create Issues for the sections. Issues are specific areas of concern you want to measure in the survey. Issues are created in a similar way as sections, by clicking the Add button in the Issues view. Issues are not shown to the user in the questionnaire form, and therefore the dialog looks a bit different, as can be seen in Figure 6.10. Only the name of the Issue is needed, not the Finnish and English Tags. There can be two types of Issues: Radio and Free. Radio means that answering the Statement that measures the Issue is done with radio buttons on a six-point scale from Strongly agree to Strongly disagree, with "I don't know" as a seventh option. Free means that the Statement has a text field for free form answers. Choose the type of your Issue from the drop down list. Also, choose which Section the Issue belongs to from the Section drop down list. Do not forget to click Save when you are ready.

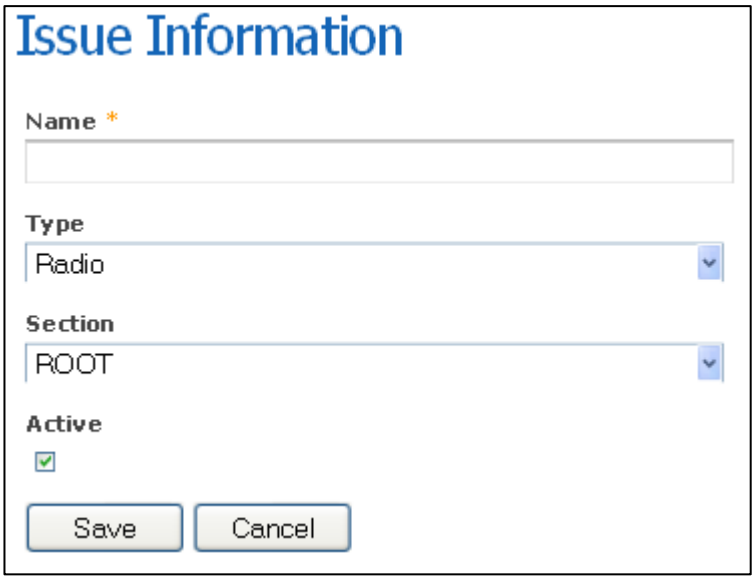


Figure 6.10: The Issue Information dialog for creating Issues

When you are ready creating the Issues you want to measure, you need to create the Statements that measure the Issue. Statements are what the users see and answer to in the questionnaire form. Statements are created from the Statements view by clicking the Add button. The dialog for creating Statements is shown in Figure 6.11. English and Finnish Statements need to be created separately. Write your Statement in the Description text field. Choose which Issue the Statement measures from the Issue drop down list. Choose the language of your Statement. The Negative tick box is needed for "inverted statements", i.e. when answering on the Agree side is not considered positive. Otherwise the logic in the HB tool is that Agree answers are interpreted as good and Disagree answers as bad and this logic is used in the calculation in the Excel report described in 6.2.3 Extracting the data from the filled in questionnaire. However, the context always dictates what actually is good or bad, and determining that is part of the analysis of the results.

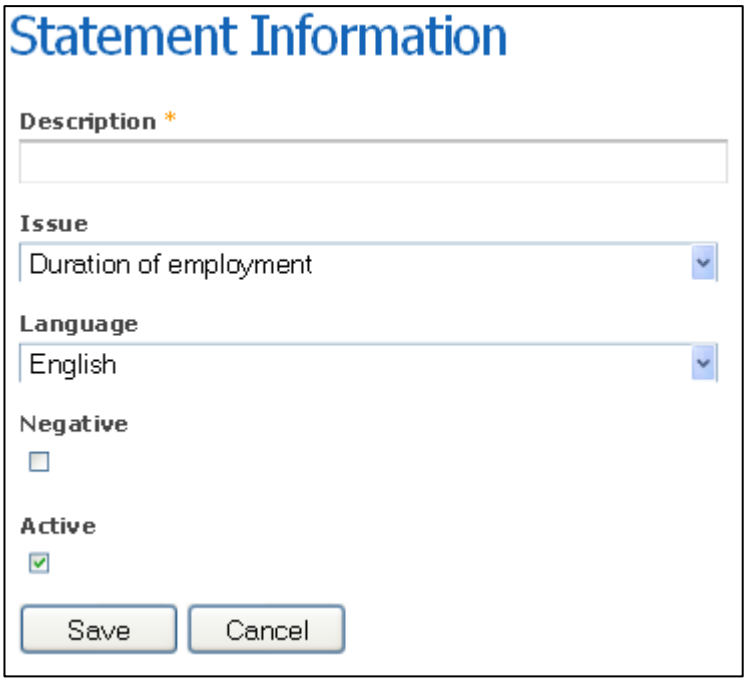


Figure 6.11: Statement Information dialog for Statement creation

When you have created the necessary Sections, Issues, and Statements for your questionnaire form, you are ready to create a new survey round. Click the Add button in the Rounds view to create a new Round. That will open the first part of the Round Information view where you choose your organization from a drop down list. When you have chosen your organization, click the Save button. This will open the second part of the Round Information view, shown in Figure 6.12. The Issues (and Section hierarchy) shown in the Issue list in Figure 6.12 are from ATMAN Default. In your case the Sections and Issues you created would be shown. In order to create the questionnaire form you need to add the Issues you want to investigate in order of appearance by choosing the Issues from the list and clicking the Add button. You can order the Issues afterwards, but it can be burdensome. The whole process of adding Issues for the questionnaire form can be painstaking, but when you have the questionnaire form ready, you can always clone it for your following rounds. When your questionnaire form is ready, you can open your round be changing its state to Started and clicking Save. You can also leave it in Draft state, but do not forget to click Save.

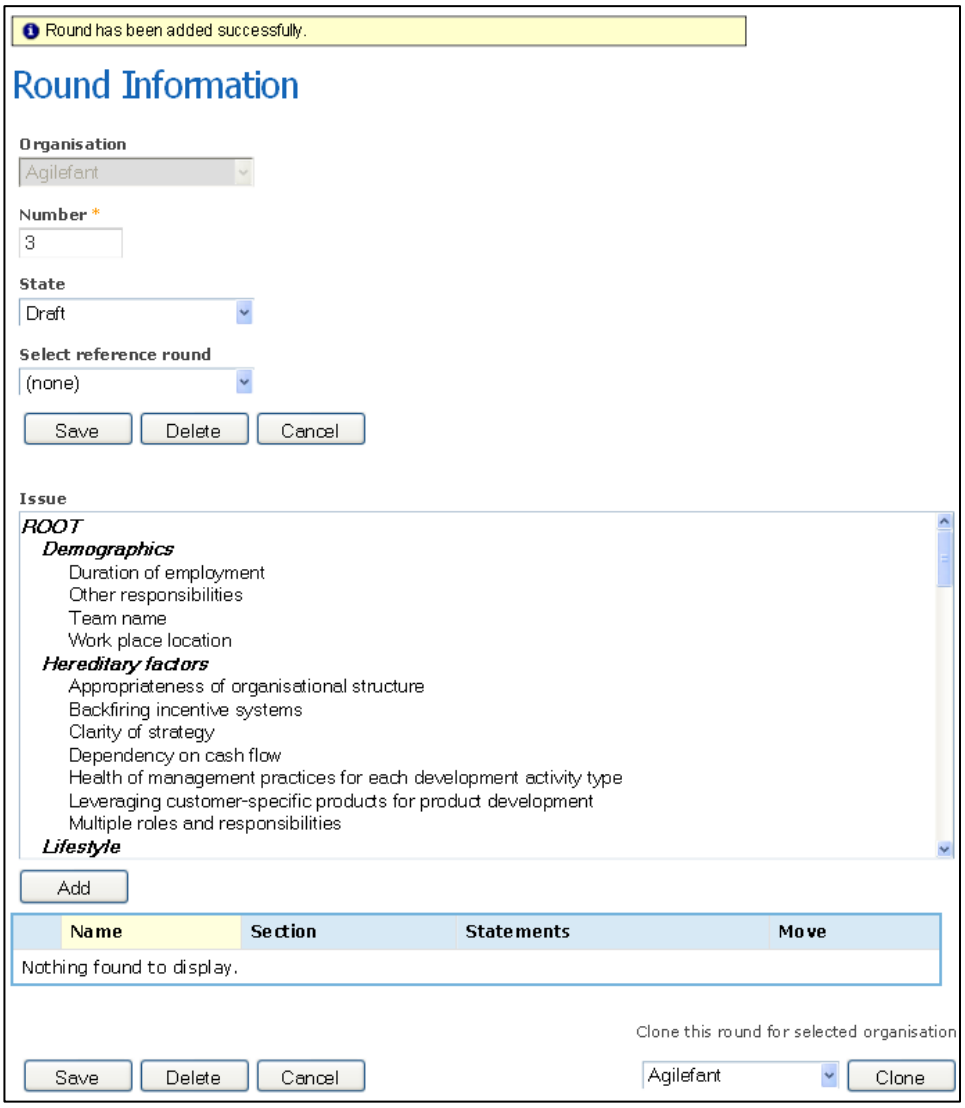


Figure 6.12: Part 2 of the Round details view

6.2.2 Maintaining the user base

Users participate in the survey by answering the questionnaire and some users are also interviewed. Users can create their own account when they log in, which is explained in 6.3 User tasks. The Administrator can also create the accounts for users, but we recommend that users do it themselves. If a user forgets his/her login information, the Administrator can view the list of users in the Users view and edit any user's information by clicking on the user, opening the User Profile view shown in Figure 6.13. The Administrator can also delete a user or render the account inactive by unchecking the Active tick box.

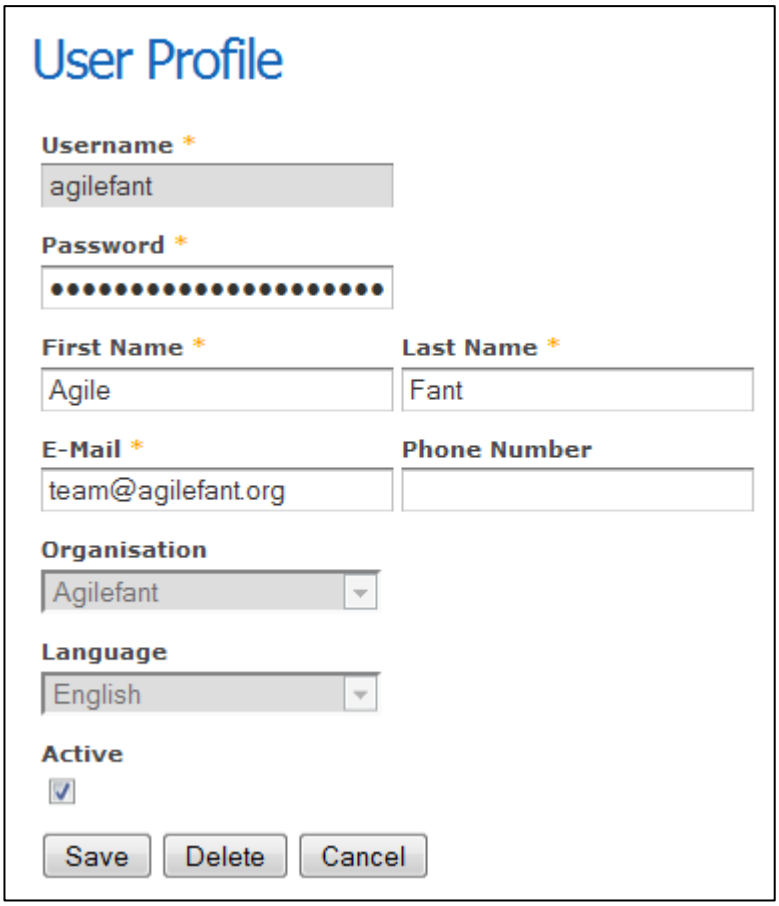


Figure 6.13: User Profile view

6.2.3 Extracting the data from the filled in questionnaire

In the Administrator front page shown in Figure 6.1 you find the Report button to the right of all Started and Closed Rounds. Clicking on the Report button saves the questionnaire data into a Microsoft Excel file. The columns of the Excel file are explained in Table 6.2. Analyzing the data is explained in Section 5.3.

Table 6.2 Explanation of the columns in the Excel data file

Column	Explanation
Section	The Section the Issue and Statements belong to.
Issue	The Issue that is investigated.
Type	Type of the Issue (Radio or Free).
Statement FI	The Statement measuring the Issue in Finnish.
Statement EN	The Statement measuring the Issue in English.
Round N: Median	N=Round number. Median of an Issue or a whole Section.
Round N: Min	The lowest value of all answers for an Issue or a whole Section.
Round N: Max	The highest value of all answers for an Issue or a whole

	Section.
Round N: Freq.	The number of answers for an Issue or a whole Section.
Username: Answer	The answer of the user as a numerical value used in the calculations. Strongly agree = 1 and Strongly disagree = 6, unless the Statement is "Negative" as explained in 6.2.1 Creating a survey. In that case the scale is inverted (Strongly agree = 6 and Strongly disagree = 1).
Username: Original answer	The Administrator can force change the user's answer, as can be seen in Figure 6.14 (the second row). If the Administrator force changes a user's answer, the changed answer is shown in the Answer column and the original answer of the user is shown in this column.
Username: User comment	The questionnaire form has a text field reserved for user comments for each Statement. If the user writes a comment it is shown in this column.
Username: Admin comment	The Administrator can comment changes or user answers, or even use this field in the Administrator view of a user's answers (Figure 6.14) to make interview notes. The Administrator's comments are shown in this column in the Excel file.

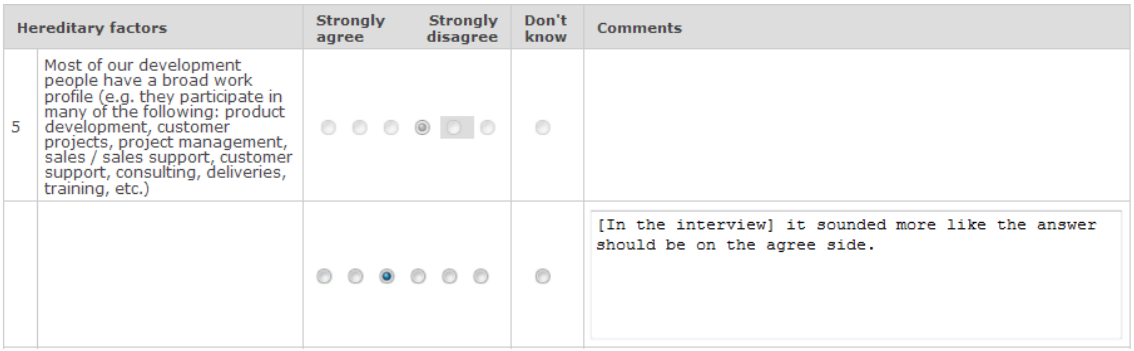


Figure 6.14: Example of Administrator view of user answers

6.3 User tasks

The user of the HB tool really has only one task, answering the Statements in the questionnaire form. To be able to do that, the user needs to sign up for an account, unless the Administrator has created an account for the user. The login dialog (Figure 6.15) gives the user the possibility to create an account by clicking the Signup link below the Login button. This opens a view similar to that in Figure 6.13 where the user fills in his/her information and presses the Signup button at the end. An e-mail is sent to the user's e-mail address, but the user can go on and directly log in to the HB tool without waiting for the e-mail.

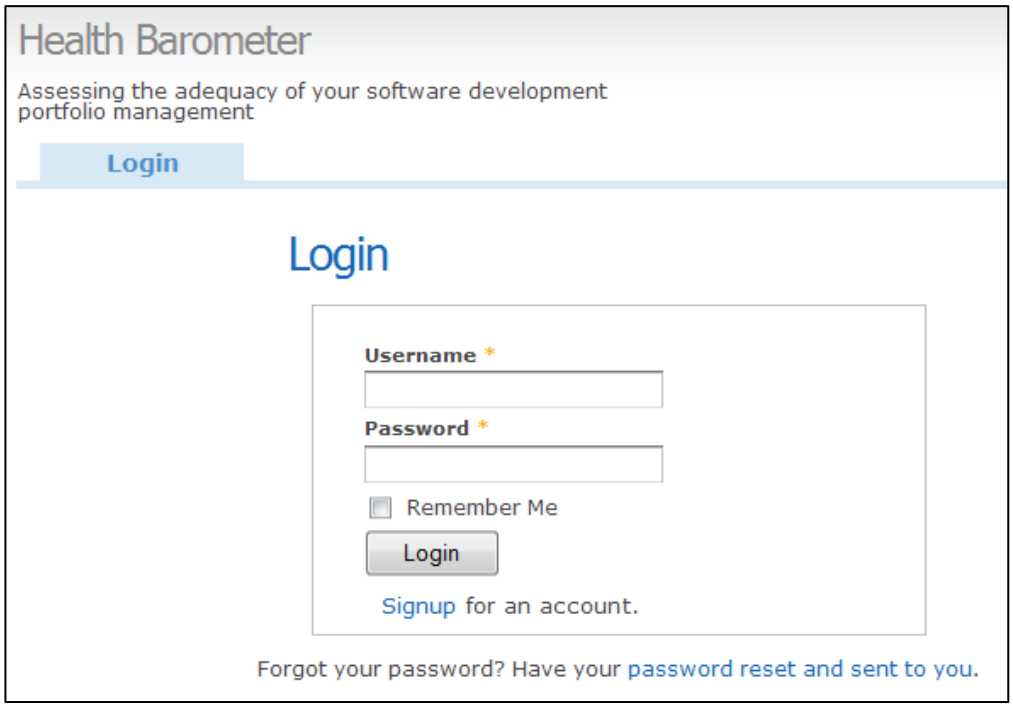


Figure 6.15: The Login dialog

When the user logs in, (s)he sees any open rounds for his/her organization, like in the example in Figure 6.16. The questionnaire is opened by clicking on the open round in question.

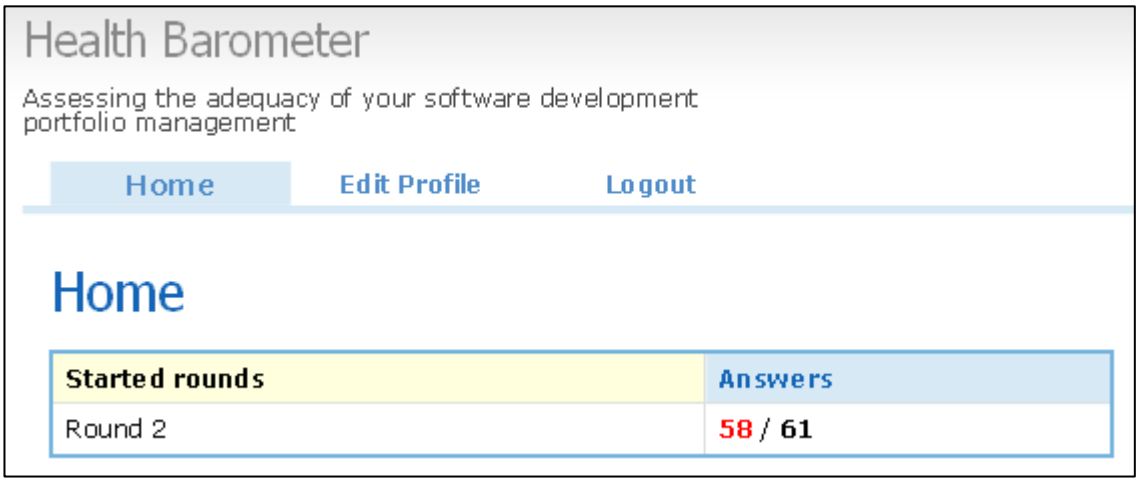
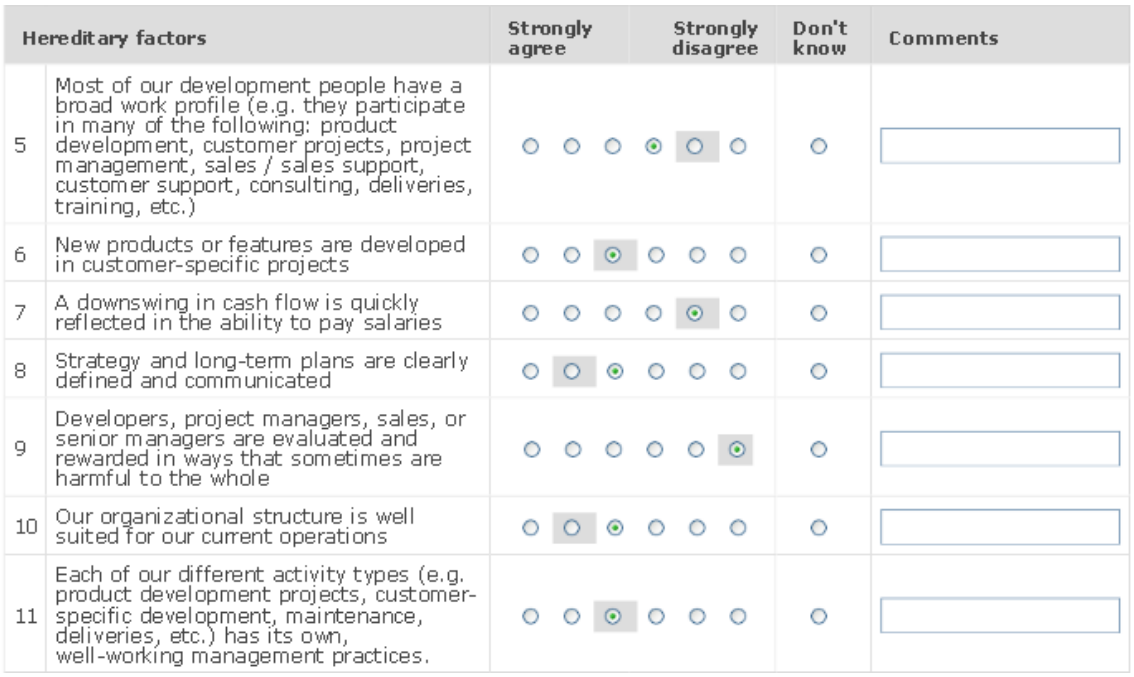


Figure 6.16: User's front page

Filling the questionnaire is like filling any other questionnaire and is not explained here, except for the one detail that is different from other questionnaires in the HB tool. The Administrator can choose to show the user his/her answers from a previous round. This is done in the Rounds Information view (Figure 6.6) by selecting the reference round. For a first round this is naturally not available. When a reference round is selected, the user sees his/her answers from the reference round (if (s)he participated in it) as grey boxes in the questionnaire form, as shown in Figure 6.17. Seeing the previous answers can bias the user's answers for the current round, but we have noticed that it lessens the noise in the an-

swers, so we like to use it. The interviews can then confirm that the answers are to the point and not too biased.



	Hereditary factors	Strongly agree				Strongly disagree		Don't know	Comments
5	Most of our development people have a broad work profile (e.g. they participate in many of the following: product development, customer projects, project management, sales / sales support, customer support, consulting, deliveries, training, etc.)	○	○	○	◉	○	○	○	
6	New products or features are developed in customer-specific projects	○	○	◉	○	○	○	○	
7	A downswing in cash flow is quickly reflected in the ability to pay salaries	○	○	○	○	◉	○	○	
8	Strategy and long-term plans are clearly defined and communicated	○	○	◉	○	○	○	○	
9	Developers, project managers, sales, or senior managers are evaluated and rewarded in ways that sometimes are harmful to the whole	○	○	○	○	○	◉	○	
10	Our organizational structure is well suited for our current operations	○	○	◉	○	○	○	○	
11	Each of our different activity types (e.g. product development projects, customer-specific development, maintenance, deliveries, etc.) has its own, well-working management practices.	○	○	◉	○	○	○	○	

Figure 6.17: Example of how the User sees his/her previous answers (grey boxes)

Appendix A: Instructions for the Health Barometer

Below is a copy of the email instructions we have used when conducting the health barometer study rounds. Feel free to adapt and utilize it as you see fit for your context.

The Health Barometer (HB) is a method developed in the ATMAN research project for assessing the adequateness of an organization's practices and processes for what we call "development portfolio management". With the HB you can see what needs to be improved next and hopefully thus make your life easier.

This transcript contains the instructions on how to answer the survey. The *DEADLINE* for answering the survey is DD.MM. at 23:59. You can answer the survey right away, it will take you some 30 minutes if you do it the first time (if you have done it before, it is a bit faster). Before starting, please take the time to quickly read through the instructions.

THOSE WHO ARE TO BE INTERVIEWED MUST ANSWER THE SURVEY BEFORE THE INTERVIEW!

Here are the instructions for completing the survey:

1) Go to http(s)://<address_of_the_survey>

2) If you have answered the survey on an earlier round, login using your existing username. You can reset your password from the login page if you don't remember it. If you have not answered the questionnaire before, you have to sign up; select your organization, and pick the language you are more familiar with (Finnish/English). You'll receive a confirmation email but you don't have to wait for it to continue. Just log in.

3) Click yourself into the open survey round (#N) by clicking on the round name and answer the survey. It will take you between 20 and 40 minutes, with the past median being around 30min.

4) Start from the responsibilities and demographics section; check those boxes that fit your responsibilities. If some of your responsibilities are missing from the possible choices, type them into the answer space for the first question after the checkboxes. Then fill in your demographic data.

All of your answers & possible comments are kept confidential, only the person(s) doing the analysis will see all the data.

5) Answer the rest of the questionnaire by evaluating how strongly you agree or disagree with the statements. If you had answered the previous round, your past answers are shown in the form as grey boxes to make comparison easier.

Answer from the perspective of your own team / unit / department -whatever is the "smallest appropriate organizational block" to evaluate the statement. Use the comments space to specify the context of your answer.

Answer realistically, without exaggerating or "tidying things up".

If you are unsure of whether you understood the statement, make a brief note in the comment field. If you understand the statement but don't know the answer, pick "I don't know". Also, rather answer "I don't know" than make a guess.

6) Remember to save your answers every now and then by clicking the button at the bottom of the form. However, please answer all of the statements, even if it takes two or more sessions!

If you are inactive for a longer period of time, there is a time-out, so remember to save your answers if you get interrupted!

The following terms are used throughout the survey:

IN FINNISH:

TEKEMISSALKKU viittaa kaikkien "kehitysporukan" (eli teknisen ja/tai tuotekehityksen henkilöstön) huomiota vaativien, meneillään sekä välittömästi suunnitteilla olevien "tekemisten" kokonaisuuteen. Esimerkkejä tyypillisistä tekemisten tyypeistä ovat tuotekehitysprojektit, ylläpito, asiakaskohtainen kehitys, toimitukset, asiakaspalvelu, koulutus, konsultointi ja myynnin tuki. Kuitenkaan tekemisen tyyppejä EIVÄT ole esim. määrittely, suunnittelu, koodaus ja testaus.

TEKEMISSALKUN HALLINTA on tekemissalkun ajan tasalla pitämisestä vastaava päätöksentekoprosessi. Tekemissalkun hallinnassa priorisoidaan tekemisiä (esim. tuotekehitysprojektit) ja päätetään niiden resursoinnista. Tekemissalkun hallinnassa päätetään myös miten äkillisesti ilmaantuvat tekemisten väliset konfliktitilanteet hoidetaan.

IN ENGLISH:

THE DEVELOPMENT PORTFOLIO is the set of ongoing and upcoming activities that require attention from the "development people" (e.g. product development and/or technical resources). Common types of development activity types are e.g. release-based product development projects, customer-specific

development, maintenance, deliveries, customer service, training, consultation, sales support, etc. However, specification, design, coding and testing are NOT types of activities we are looking for here.

DEVELOPMENT PORTFOLIO MANAGEMENT is the decision process for updating and revising the development portfolio. In development portfolio management, development activities (e.g. projects) are prioritized and resourced Development portfolio management is also responsible for appropriately resourcing the handling of suddenly emerging urgencies.

If you encounter problems in answering the questionnaire, whether technical or otherwise, do not hesitate to contact N.N. (person responsible for the survey).

The results of the survey & the interviews will be disseminated on DD.MM. N.N. will inform you of the exact time and place.

Best regards,

N.N.

Appendix B: ATMAN Default Questionnaire in English

Survey - Round 4

Agilefant - agilefant (Agile Fant)

Responsibilities (check multiple when appropriate)

- ☐ Account manager
- ☐ Application development (e.g. client)
- ☐ Architect
- ☐ Business
- ☐ Company board (Hallitus)
- ☐ Customer service
- ☐ Delivery
- ☐ Developer
- ☐ Human resources
- ☐ Internal customer
- ☐ Junior
- ☐ Lead developer
- ☐ Maintenance
- ☐ Manager (C*O-level)
- ☐ Manager (middle)
- ☐ Manager (VP-level)
- ☐ Platform development (e.g. server)
- ☐ Portfolio manager
- ☐ Process improvement
- ☐ Product management
- ☐ Product management (solution-level)
- ☐ Product management (technical)
- ☐ Product owner
- ☐ Program manager
- ☐ Project management
- ☐ Quality manager
- ☐ Requirements engineer
- ☐ Sales
- ☐ Scrum master
- ☐ Senior
- ☐ Team leader
- ☐ Testing

Demografic information		Strongly agree	Strongly disagree	Don't know	Comments
1	What other responsibilities do you have, if any?	Answer belowor *I don't know*			
2	What is the name of your team?	Answer belowor *I don't know*			
3	Where is your work place located (which site, floor, etc.)?	Answer belowor *I don't know*			
4	How long have you been employed in the company?	Answer belowor *I don't know*			
5	Our business is profitable	○ ○ ○ ○ ○ ○		○	
Hereditary factors		**Strongly agree**	**Strongly disagree**	**Don't know**	**Comments**
6	Most of our development people have a broad work profile (e.g. they participate in many of the following: product development, customer projects, project management, sales / sales support, customer support, consulting, deliveries, training, etc.)	○ ○ ○ ○ ○ ○		○	
7	New products or features are developed in customer-specific projects	○ ○ ○ ○ ○ ○		○	
8	A downswing in cash flow is quickly reflected in the ability to pay salaries	○ ○ ○ ○ ○ ○		○	
9	Strategy and long-term plans have been clearly defined	○ ○ ○ ○ ○ ○		○	
10	Strategy and long-term plans have been clearly communicated	○ ○ ○ ○ ○ ○		○	
11	Developers, project managers, sales, or senior managers are evaluated and rewarded in ways that are harmful to the whole	○ ○ ○ ○ ○ ○		○	
12	Our organisational structure supports our current operations	○ ○ ○ ○ ○ ○		○	
13	Each of our different activity types (e.g. product development projects, customer-specific development, maintenance, deliveries, etc.) has its own practices, that work	○ ○ ○ ○ ○ ○		○	

	Structure of the development portfolio	Strongly agree	Strongly disagree	Don't know	Comments
14	We have identified the different types of activities development people spend their time on (e.g. product development projects, customer-specific development, maintenance, deliveries, etc.)	○ ○ ○ ○ ○ ○		○	
15	Business people are able to see the 'big picture' of ongoing activities (a.k.a. the development portfolio)	○ ○ ○ ○ ○ ○		○	
16	Development people are able to see the 'big picture' of ongoing activities (a.k.a. the development portfolio)	○ ○ ○ ○ ○ ○		○	
17	I understand how much time, from a business perspective, I should spend on different types of activities	○ ○ ○ ○ ○ ○		○	
18	We have criteria for prioritising our ongoing development activities	○ ○ ○ ○ ○ ○		○	
19	I understand the priorities between ongoing activities (e.g. project X vs. project Y, project X vs. support request Z, etc.)	○ ○ ○ ○ ○ ○		○	
20	I understand the dependencies of the ongoing activities	○ ○ ○ ○ ○ ○		○	
	Process and practices of development portfolio management	**Strongly agree**	**Strongly disagree**	**Don't know**	**Comments**
21	All the ongoing and immediately upcoming activities that require attention from the developers are managed as an explicit portfolio	○ ○ ○ ○ ○ ○		○	
22	We have defined who are responsible for development portfolio management	○ ○ ○ ○ ○ ○		○	
23	It is clear who should participate in development-related decision making in different situations (e.g. in the middle of a project, when an urgent maintenance request arrives, when making an offer, when deploying a product, etc.)	○ ○ ○ ○ ○ ○		○	
24	We actively reflect the content of the development portfolio to the company's strategy	○ ○ ○ ○ ○ ○		○	
25	In decision making we mainly consider individual activities and do not take the "big picture" into account	○ ○ ○ ○ ○ ○		○	

Multitasking		Strongly agree	Strongly disagree	Don't know	Comments
26	How many different activities (product development projects, customer projects, etc.) are currently ongoing in your company?	Answer belowor *I don't know*			
27	How many different activities (product development projects, customer projects, etc.) are currently ongoing that you are involved in?	Answer belowor *I don't know*			
28	In addition to my main responsibility, I also have other, time-demanding responsibilities	○ ○ ○ ○ ○ ○		○	
29	We have too many parallel ongoing activities	○ ○ ○ ○ ○ ○		○	
30	A single person is usually assigned to only one activity (e.g. a project) at the same time	○ ○ ○ ○ ○ ○		○	
31	We complete one thing at a time and don't shift our attention from one incomplete task to another	○ ○ ○ ○ ○ ○		○	
Fire fighting		**Strongly agree**	**Strongly disagree**	**Don't know**	**Comments**
32	"Fire fighting" describes our work in practice	○ ○ ○ ○ ○ ○		○	
33	Resources are being shifted from one activity (e.g. a project) to another regardless of previously agreed assignments	○ ○ ○ ○ ○ ○		○	
34	Changes in resourcing for one activity (e.g. a project) cause uncontrolled changes in other activities	○ ○ ○ ○ ○ ○		○	
35	Resource commitments are too rigid for leveraging suddenly emerging opportunities	○ ○ ○ ○ ○ ○		○	
Overcommitment		**Strongly agree**	**Strongly disagree**	**Don't know**	**Comments**
36	I work overtime	○ ○ ○ ○ ○ ○		○	
37	When planning product releases or making offers, we consider how to resource the work in practice	○ ○ ○ ○ ○ ○		○	
38	New activities (e.g. projects) are launched too often	○ ○ ○ ○ ○ ○		○	
39	Our employees have too much to do and quality of work suffers from it	○ ○ ○ ○ ○ ○		○	
40	We have enough resources in proportion to the amount of work	○ ○ ○ ○ ○ ○		○	

Decision making		Strongly agree				Strongly disagree	Don't know	Comments
41	Activities (e.g. projects) are never killed	○	○	○	○	○	○	○	
42	If time runs out, developers resolve by themselves what can be left undone	○	○	○	○	○	○	○	
43	The real status of activities is known in development portfolio -level decision making	○	○	○	○	○	○	○	
44	The priority ranking of activities changes constantly	○	○	○	○	○	○	○	
45	Management reacts to problems detected in activities (e.g. projects) too late	○	○	○	○	○	○	○	
46	Senior management is actively involved in portfolio-level decision making	○	○	○	○	○	○	○	
47	The dialogue between Business and Development people works	○	○	○	○	○	○	○	
Strategic alignment and portfolio balance		**Strongly agree**				**Strongly disagree**	**Don't know**	**Comments**
48	Ongoing activities are in alignment with the company's strategy	○	○	○	○	○	○	○	
49	Ongoing activities are essential to our business	○	○	○	○	○	○	○	
50	We have a sufficient amount of development projects that incrementally improve existing products or services	○	○	○	○	○	○	○	
51	We have a sufficient amount of product or service development projects that aim for new business	○	○	○	○	○	○	○	
Slipping schedules		**Strongly agree**				**Strongly disagree**	**Don't know**	**Comments**
52	Ongoing activities are behind schedule	○	○	○	○	○	○	○	
53	Progress of ongoing activities is reported optimistically	○	○	○	○	○	○	○	
Change in performance		**Strongly agree**				**Strongly disagree**	**Don't know**	**Comments**
54	From a business viewpoint, development performs its duties well	○	○	○	○	○	○	○	
55	Our capability to produce high-quality software has improved during the past year	○	○	○	○	○	○	○	
Perceived improvement needs		**Strongly agree**				**Strongly disagree**	**Don't know**	**Comments**
56	We should invest more in improving the practices of individual activities (e.g. project mgmt., team practices, deployment processes, sales processes, customer support, etc.)	○	○	○	○	○	○	○	
57	We should invest more in improving development portfolio management (e.g. prioritizing activities, linking strategy with daily work, structuring the development portfolio, etc.)	○	○	○	○	○	○	○	

Appendix C: ATMAN Default Questionnaire in Finnish

Demografiatiedot		Vahvasti samaa mieltä	Vahvasti eri mieltä	En tiedä	Kommentit
1	Mitkä ovat muut vastuusi, jos niitä on?	Kirjoita vastaus alle tai *En tiedä*			
2	Missä tiimissä teet työtä?	Kirjoita vastaus alle tai *En tiedä*			
3	Missä työpisteesi sijaitsee (mikä toimisto, kerros, jne.)?	Kirjoita vastaus alle tai *En tiedä*			
4	Kuinka pitkään olet ollut yrityksessä töissä?	Kirjoita vastaus alle tai *En tiedä*			
5	Liiketoimintamme on kannattavaa	○ ○ ○ ○ ○ ○		○	
Perintötekijät		**Vahvasti samaa mieltä**	**Vahvasti eri mieltä**	**En tiedä**	**Kommentit**
6	Suurella osalla kehitysporukastamme on laaja tehtävänkuva (esim. osallistuu moniin seuraavista: tuotekehitys, asiakasprojektit, projektien vetäminen, myynnin tuki / myynti, asiakastuki, konsultointi, toimitukset, kouluttaminen jne.)	○ ○ ○ ○ ○ ○		○	
7	Uudet tuotteet tai ominaisuudet kehitetään asiakaskohtaisissa projekteissa	○ ○ ○ ○ ○ ○		○	
8	Notkahdus kassavirrassa heijastuu nopeasti palkanmaksukykyyn	○ ○ ○ ○ ○ ○		○	
9	Strategia ja pitkän tähtäimen suunnitelmat on määritelty selkeästi	○ ○ ○ ○ ○ ○		○	
10	Strategia ja pitkän tähtäimen suunnitelmat on kommunikoitu selkeästi	○ ○ ○ ○ ○ ○		○	
11	Kehittäjien, projektipäälliköiden, myynnin tai johdon suoriutumista arvioidaan ja palkitaan tavoilla, joista on haittaa kokonaisuudelle	○ ○ ○ ○ ○ ○		○	
12	Organisaatiorakenteemme tukee nykyistä toimintaamme	○ ○ ○ ○ ○ ○		○	
13	Jokaisella eri tyyppisellä tekemisellä (esim. tuotekehitysprojektit, asiakaskohtainen kehitys, ylläpito, toimitukset, jne.) on omat toimivat käytäntönsä	○ ○ ○ ○ ○ ○		○	

Tekemissalkun jäsennys		Vahvasti samaa mieltä				Vahvasti eri mieltä		En tiedä	Kommentit
14	Olemme tunnistaneet eri tyyppiset tekemiset joihin kehitysporukka käyttää aikaa (esim. tuotekehitysprojektit, asiakaskohtainen kehitys, ylläpito, toimitukset, jne.)	○	○	○	○	○	○	○	
15	Liiketoiminnasta vastaavat hahmottavat meneillään olevien tekemisten kokonaisuuden (eli tekemissalkun)	○	○	○	○	○	○	○	
16	Kehitysporukka hahmottaa meneillään olevien tekemisten kokonaisuuden (eli tekemissalkun)	○	○	○	○	○	○	○	
17	Ymmärrän miten ajankäyttöni tulisi liiketoiminnan näkökulmasta jakautua eri tyyppisten tekemisten kesken	○	○	○	○	○	○	○	
18	Meillä on kriteeristö meneillään olevien tekemisten priorisointiin	○	○	○	○	○	○	○	
19	Ymmärrän meneillään olevien tekemisten keskinäisen tärkeysjärjestyksen (esim. projekti X vs. projekti Y, projekti X vs. ylläpitopyyntö Z, jne.)	○	○	○	○	○	○	○	
20	Ymmärrän meneillään olevien tekemisten riippuvuudet	○	○	○	○	○	○	○	
Salkunhallinnan toimintatavat		**Vahvasti samaa mieltä**				**Vahvasti eri mieltä**		**En tiedä**	**Kommentit**
21	Kaikkia meneillään olevia sekä välittömästi suunnitteilla olevia, kehitysporukan huomiota vaativia tekemisiä hallitaan kokonaisuutena, eli tekemissalkkuna	○	○	○	○	○	○	○	
22	Olemme määritelleet ketkä vastaavat tekemissalkun hallinnasta	○	○	○	○	○	○	○	
23	On selvää keiden tulisi missäkin tilanteissa osallistua ohjelmistokehitystä koskevaan päätöksentekoon (esim. kesken projektin, kiireellisen ylläpitopyynnön sattuessa, tarjousta tehdessä, käyttöönotossa, jne.)	○	○	○	○	○	○	○	
24	Peilaamme aktiivisesti tekemissalkun sisältöä yrityksen strategiaan	○	○	○	○	○	○	○	
25	Päätöksenteossa tarkastellaan lähinnä yksittäisiä tekemisiä eikä kokonaiskuvaa huomioida	○	○	○	○	○	○	○	

Moniajo		Vahvasti samaa mieltä	Vahvasti eri mieltä	En tiedä	Kommentit
26	Kuinka monta eri tekemistä (tuotekehitysprojektit, asiakasprojektit, jne.) on yrityksessänne tällä hetkellä meneillään?	Kirjoita vastaus alle tai *En tiedä*			
27	Kuinka monta eri tekemistä (tuotekehitysprojektit, asiakasprojektit, jne.) on tällä hetkellä meneillään, joissa itse olet mukana?	Kirjoita vastaus alle tai *En tiedä*			
28	Varsinaisen päätehtäväni lisäksi minulla on myös muita, aikaa vaativia vastuita	○ ○ ○ ○ ○ ○		○	
29	Meillä on käynnissä samanaikaisesti liian monta tekemistä	○ ○ ○ ○ ○ ○		○	
30	Yksittäinen henkilö on tavallisesti kiinnitetty vain yhteen tekemiseen (esim. projektiin) kerrallaan	○ ○ ○ ○ ○ ○		○	
31	Teemme asian kerrallaan valmiiksi emmekä siirrä huomiota keskeneräisestä tehtävästä toiseen	○ ○ ○ ○ ○ ○		○	
Tulipaloherkkyys		**Vahvasti samaa mieltä**	**Vahvasti eri mieltä**	**En tiedä**	**Kommentit**
32	"Tulipalojen sammuttaminen" kuvaa toimintaamme käytännössä	○ ○ ○ ○ ○ ○		○	
33	Resursseja siirrellään tekemisistä toisiin aiemmista kiinnityksistä huolimatta	○ ○ ○ ○ ○ ○		○	
34	Muutokset resursoinnissa yhteen tekemiseen (esim. projektiin) aiheuttavat hallitsemattomia muutoksia muihin tekemisiin	○ ○ ○ ○ ○ ○		○	
35	Resursointi on niin jäykkää, ettei yllättäviä mahdollisuuksia ja tilaisuuksia kyetä hyödyntämään	○ ○ ○ ○ ○ ○		○	
Ylikuormitus		**Vahvasti samaa mieltä**	**Vahvasti eri mieltä**	**En tiedä**	**Kommentit**
36	Teen ylitöitä	○ ○ ○ ○ ○ ○		○	
37	Tuotejulkaisuja suunnitellessa tai myyntiä tehdessä huomioidaan miten tekemiset tullaan käytännössä resursoimaan	○ ○ ○ ○ ○ ○		○	
38	Uusia tekemisiä (esim. projekteja) käynnistetään liian usein	○ ○ ○ ○ ○ ○		○	
39	Työntekijöillämme on liikaa tekemistä ja työn laatu kärsii siitä	○ ○ ○ ○ ○ ○		○	
40	Resursseja on riittävästi suhteessa tekemisen määrään	○ ○ ○ ○ ○ ○		○	

Päätöksenteko		Vahvasti samaa mieltä					Vahvasti eri mieltä	En tiedä	Kommentit
41	Tekemisiä (esim. projekteja) ei koskaan keskeytetä lopullisesti	○	○	○	○	○	○	○	
42	Jos aika loppuu kesken, kehittäjät ratkaisevat keskenään mitä voi jättää tekemättä	○	○	○	○	○	○	○	
43	Salkkutason päätöksenteossa tiedetään tekemisten todelliset tilanteet	○	○	○	○	○	○	○	
44	Tekemisten tärkeysjärjestys muuttuu jatkuvasti	○	○	○	○	○	○	○	
45	Johto reagoi tekemisissä (esim. projekteissa) havaittuihin ongelmiin liian myöhään	○	○	○	○	○	○	○	
46	Ylin johto osallistuu aktiivisesti tekemissalkkutason päätöksentekoon	○	○	○	○	○	○	○	
47	Liiketoiminnasta vastaavien ja kehitysporukan välinen vuoropuhelu toimii	○	○	○	○	○	○	○	
Tekemisten suhde strategiaan ja tekemissalkun tasapaino		**Vahvasti samaa mieltä**					**Vahvasti eri mieltä**	**En tiedä**	**Kommentit**
48	Meneillään olevat tekemiset ovat linjassa yrityksen strategian kanssa	○	○	○	○	○	○	○	
49	Meneillään olevat tekemiset ovat liiketoiminnan kannalta keskeisiä	○	○	○	○	○	○	○	
50	Meillä on riittävästi olemassaolevien tuotteiden tai palveluiden jatkokehittämiseen keskittyviä projekteja	○	○	○	○	○	○	○	
51	Meillä on riittävästi uuteen liiketoimintaan tähtääviä tuote- tai palvelukehitysprojekteja	○	○	○	○	○	○	○	
Aikataulujen pito		**Vahvasti samaa mieltä**					**Vahvasti eri mieltä**	**En tiedä**	**Kommentit**
52	Meneillään olevat tekemiset ovat jäljessä aikataulusta	○	○	○	○	○	○	○	
53	Tekemisten valmiusastetta raportoidaan optimistisesti	○	○	○	○	○	○	○	
Muutos suorituskyvyssä		**Vahvasti samaa mieltä**					**Vahvasti eri mieltä**	**En tiedä**	**Kommentit**
54	Liiketoiminnan näkökulmasta ohjelmistokehitys suoriutuu tehtävistään hyvin	○	○	○	○	○	○	○	
55	Kykymme tuottaa korkealaatuista ohjelmistoa on parantunut viimeisen vuoden aikana	○	○	○	○	○	○	○	
Oma näkemys kehitystarpeesta		**Vahvasti samaa mieltä**					**Vahvasti eri mieltä**	**En tiedä**	**Kommentit**
56	Yksittäisten tekemisten käytäntöjen kehittämiseen pitäisi panostaa enemmän (esim. projektinhallinta, tiimikäytännöt, toimitusprosessit, myynnin prosessit, asiakastuki, jne.)	○	○	○	○	○	○	○	
57	Tekemissalkun hallinnan kehittämiseen pitäisi panostaa enemmän (esim. tekemisten priorisointi, strategian ja päivittäisen työn linkitys, tekemissalkun jäsentäminen, jne.)	○	○	○	○	○	○	○	

PART III: FRAMEWORK AND PRACTICES FOR AGILE PRODUCT AND PORTFOLIO MANAGEMENT

Part III of this book presents our framework for agile product and portfolio management (chapters 7-9) and selected practices regarding areas of agile product and portfolio management that have proved challenging in practice (chapters 10-12). Chapter 7 discusses how long-term product and release planning should be understood in the context of agile software development, and describes the ATMAN framework for linking strategy with action. In Chapter 8 we explain the levels of portfolio management for governing an agile enterprise, present a framework for portfolio decision-making in agile software development based on these levels as well as a series of steps for setting up agile portfolio management. Chapter 9 provides an in-depth discussion of the complexities related to proper backlog management in managing an agile development portfolio. Here, we also relate our framework to software companies where multi-tasking and consequent fire-fighting have become the de-facto management process. Chapter 10 explains how a company reorganized its product management process to better link it with the R&D who used Scrum. Chapter 11 discusses the joint release planning method for performing release planning when there are many teams working on the same product. Chapter 12 presents the basics of Kanban and explains how having multiple teams working on multiple products can be managed using a Kanban board. Chapter 13 rounds up part III by summarizing the most important requirements for backlog management tool support for making product and portfolio management work with agile software development.

Chapter 7: Agile Product Management

Jarno Vähäniitty

If fast-paced development is not properly managed, there is a danger that fragmented results are produced and the big picture of the ongoing work and its contribution to the business goals or the company's overall strategy becomes unclear. To realize the benefits from an agile software development process for the entire organization, agile methods should be extended from addressing individual development projects to long-term product and release planning and portfolio management.

In this chapter we explore existing work and definitions of release planning (Section 7.1) and roadmapping (Section 7.2), and explain how they should be understood in the context of agile software development (Section 7.3). Section 7.3 also explains why retaining the trace between the high-level goals expressed in long-term plans (e.g. business goals) and short-term objectives (e.g. individual user stories and tasks) is what can make (but more commonly break) the connection between product management and agile software development, and describes the *ATMAN framework for linking strategy with action*.

7.1 What is release planning?

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

While release planning has attracted attention among software engineering researchers, most of the research treats release planning (Akker et al. 2008, Chatzipetrou et al. 2010, Ngo-The & Ruhe 2009, Mc Elroy & Ruhe 2010, Al-Emran, Pfahl & Ruhe 2010) essentially as an optimization problem (Kittlaus & Clough 2009). Most of the existing work on release planning has focused on developing model-based approaches designed for a situation where there is a single product/service offering with a set of possible features to be selected from (Svahn-

berg et al. 2010). These features are assumed to have been elaborated to the degree that their development cost and business value can be reasonably estimated. Also, it is assumed that relevant stakeholders are readily available to familiarize themselves with the requirements and vote on them.

Unfortunately, one or more of the above listed assumptions do not hold in practice (Svahnberg et al. 2010, Lehtola 2006), rendering the optimization models as near-useless. For example, the degree of up-front requirements elaboration needed by the approaches is often not feasible or even desirable (Larman & Vodde 2010, Poppendieck & Poppendieck 2009). Also, in practice requirements are not prioritized as a one-off activity, but in multiple phases of development, with each phase involving different kinds of decision-making (Lehtola 2006). Furthermore, there are often requirements from more than a single product/service offering for the development staff to work on (Rothman 2007, Dobson 1999, Rothman 2009).

Overall, existing systematic algorithmic approaches to planning the future development steps of a particular product/service offering often seem to have unfortunately little applicability to the actual decision-making problem faced by practitioners (Ivarsson & Gorschek 2009). Thus, it is hardly surprising that most approaches to release planning have not been validated in an industrial setting (Svahnberg et al. 2010).

Rather than further devising models for "optimizing" the contents of upcoming releases, this book takes the stand that it should first be understood how the roadmapping and release planning processes actually manifest themselves in agile software development. This is discussed in Section 7.3 (*Linking agile with long-term product and release planning*). A more detailed explanation of how release planning should be conducted when there is more than a single team working on the same solution is given in Chapter 11: Scaling Up Agile Release Planning.

7.2 What is roadmapping?

Product roadmapping (or simply roadmapping) is a common metaphor for planning the allocation of development capacity and the use of technology as well as their relationships over a period of time. The process of roadmapping should identify, evaluate and select strategic alternatives for achieving desired objectives (Kostoff & Schaller 2001). The resulting roadmaps summarize and communicate the results of key business decisions (DeGregorio 2000). Thus, the roadmaps' implementability is at least as important as their possible strategic value (Kostoff & Schaller 2001).

There is little research literature on software product roadmapping (Fleury et al. 2006). Thus, in order to summarize existing understanding on software product roadmapping, we examine what two recent books by recognized expert practi-

tioners, one from the perspective of software product management (Kittlaus & Clough 2009) and the other from the agile software development movement (Pichler 2010) say of roadmaps and roadmapping.

The discussion below is structured in terms of the *definition* of a roadmap, *what should be included* in a roadmap, the *purpose* of roadmapping, the *timeframe* for roadmapping, how often the roadmaps should be *updated,* as well as *who should be involved* in roadmapping.

Definition: A software product roadmap is a planning artifact showing an overview of how a product is likely to evolve over a strategic timeframe of six months (Kittlaus & Clough 2009, Pichler 2010) to up to five years (Kittlaus & Clough 2009).

What should be included: A product roadmap should state the upcoming releases, their projected launch dates, the target customers, their needs, and (up to) top 5 features (Pichler 2010). Usually, important dependencies on other products or technologies are also depicted (Kittlaus & Clough 2009). According to (Pichler 2010), the product roadmap should *be simple and focused on the essentials, as the details will emerge and be captured in the product backlog*.

The purpose of roadmapping: Pichler (2010) simply states that the roadmap facilitates the dialogue between the Scrum team and the stakeholders. It allows the organization to coordinate the development and launch of related products, for instance a product line or a product portfolio. Kittlaus and Clough elaborate on this by stating that the main purpose of a roadmap is to give direction both internally and externally:

Internally it shows the relationship between product plans and financial forecasts and the major themes and requirements governing the plan. It indicates if the product will provide continuing career opportunities for employees who work on it, be they developers, sales, or support specialists. The roadmap is important for a product manager in order to reach agreement within his company regarding the longer-term direction and priorities, and for the roadmap's useful effects described above. Externally the roadmap plays an important role in demonstrating the viability of a product as well. Often bigger potential customers are willing to sign non-disclosure agreements in order to see a product roadmap before they make a significant investment decision. Similarly, market analysts mostly base their judgment on a convincing story about a product's future expressed in the roadmap. (Kittlaus & Clough 2009, p. 77)

The timeframe: 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, and *the next 6-12 months rather than next two to three years* according to Pichler (2010) in a roadmap are *more or less*

reliable, though still subject to slippages caused by development (Kittlaus & Clough 2009) or *changes in direction* (Pichler 2010). Pichler states that 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). However, the outer years can also be considered a formal way of saying that the vendor has a long-term commitment to the product (Kittlaus & Clough 2009).

Updating the roadmap: According to Pichler (2010), a product roadmap states how the developing organization believes the product is likely to evolve based on the current understanding of the market. Roadmaps *are living documents that evolve and change*. The product roadmap should be created once the product has been successfully introduced into the marketplace. Kittlaus and Clough (2009) say that the roadmap *is usually updated as part of the corporate planning cycle*, which differs from agile methods' notion of continuous planning (Shalloway, Beaver & Trott 2009).

Who will create & update the roadmap? Pichler (2010) denotes that the relevant people to create and update the product roadmap include at least the product owner and the development team, but it might also involve the person in charge of the product portfolio and representatives from other product development teams and product management.

7.3 Linking agile with long-term product and release planning

Overall, in much of the literature on agile software development, the complexities of release planning and roadmapping are simply crammed into *using a product backlog* and considered as something the product owner gets done.

Thus, it is of essence to understand how the product backlog should function in long-term product and release planning. In the first part (Section 7.3.1 *Roadmapping, release planning and the product backlog*), we explain how the product backlog relates to long-term product and release planning, or more specifically, roadmapping and release planning, and provide agile-compatible definitions of these concepts. Then, we discuss the prevailing dichotomy of whether one should retain the trace of how smaller work items have been split from the larger ones, and explain why retaining the trace is indeed important (Section 7.3.2 *Splitting work items and traceability*). We end the section in presenting our framework for linking long-term product and business goals with daily tasks (Section 7.3.3 *From strategy to action and back again*).

7.3.1 Roadmapping, release planning and the product backlog

The *product backlog*[17] is a list of all the work that currently can be seen as potentially useful to perform in order for an offering to succeed and prosper. It contains all the features, functions, technologies, enhancements, and bug fixes that constitute the changes that should and could be made to the product for future releases. The work items are ordered sequentially according to priority, with the topmost items being more urgent and/or important than those beneath them. The product vision drives this prioritization by describing the long-term objective(s) for the offering (see Figure 7.1).

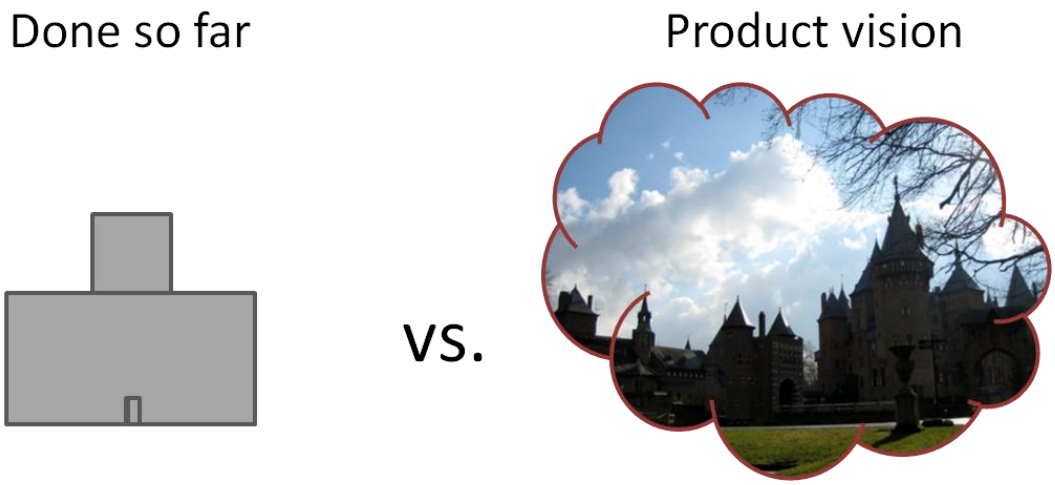


Figure 7.1: Planning is driven by comparing what's been done so far to an up-to-date product vision

The product backlog can be viewed as consisting of four sections (see Figure 7.2 below). The topmost section, *iteration backlog*, contains those work items that have been committed to for the ongoing development iteration and the tasks that are needed to get the work items done. In the case of multiple teams working in parallel on the same product, each team has its own iteration backlog. Beneath the iteration cut-off line, the next section is the *release backlog*, containing those work items that are currently thought as to be included in the first upcoming release of the product. The sections beneath the cut-off line of the ongoing release outline the contents of foreseeable future releases. At the bottom, there may be many more work items that have been thought of but so far have not been seen as crucial enough to attend to in the foreseeable future.

[17] The description in this chapter has been compiled by Jarno Vähäniitty from multiple sources (Schwaber & Beedle 2002, Shalloway, Beaver & Trott 2009, Leffingwell forthcoming 2011, Vlaanderen et al. 2009, Pichler 2010, Schiel 2009, Galen 2009, Cohn 2010, Leffingwell 2007, Schwaber & Sutherland 2010). It is based on the Scrum framework, as Scrum is the most well-known and widely adopted framework for managing agile software development (Krebs 2008).

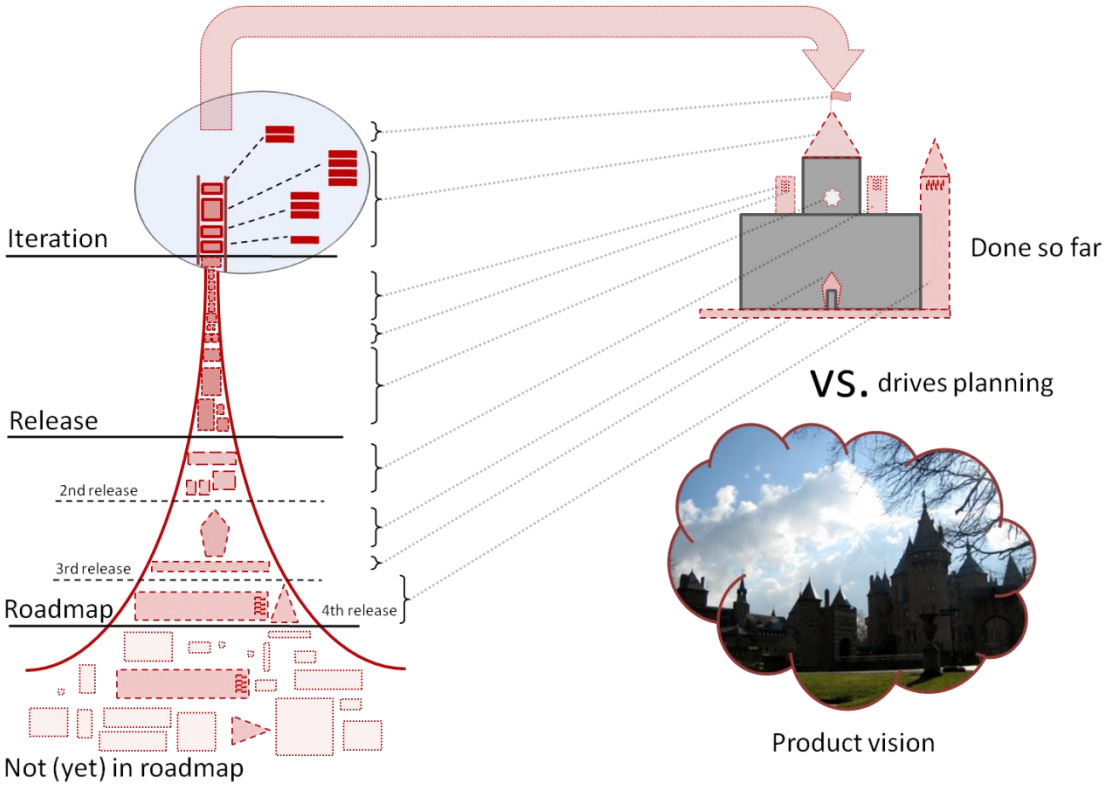


Figure 7.2: Long-term product and release planning and the product backlog

The higher the items are in the product backlog, the clearer and more detailed they should be in terms of their description, effort estimates and relative priorities. For example, the work items committed to for the iteration are in absolute rank-order, and the actual tasks needed to get them done have been fleshed out and the effort left for completing them is estimated in man-hours. The items in the section regarding the ongoing release as well as the roadmap are estimated in more abstract terms that typically have no clear connection to calendar time, for example story points or T-shirt sizes (e.g. XS, S, M, L, XL, XXL). The work items in the bottom sections of the product backlog are often large and vague. However, they can also be small and detailed, such as in the case of a laborious but basically simple bug fix that needs to be done at some point but which has not been seen as crucial to attend to just yet.

Although the product backlog is defined to be a prioritized *list*, only the priority order of the topmost items should in practice be considered to be thought through and absolute (even this tends to change). It is seldom worth the effort to prioritize the items further down the backlog – for example, beyond the current release – with the same degree of accuracy as those items that are currently being worked on. Figure 7.2 above depicts this with having items in the lower sections of the backlog on the same level horizontally.

Based on the above analysis, we define *release planning*, *roadmap*, and *roadmapping* as follows:

Release planning refers to planning and refining the contents of the immediately upcoming (ongoing) release.

The **roadmap** 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. The roadmap shows the release dates, the planned features (possibly with related epics and strategic product themes), their sizes in story points and the planned resource usage. 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 means grooming the product backlog so that it reflects the current understanding of the relevant stakeholders in terms of foreseeable future releases and the features planned to be included in them.

7.3.2 Splitting work items and traceability

As development proceeds, the ideal is to have at least the top sections of the product backlog (see Figure 7.2) constantly up-to-date and ready to feed development with detailed-enough user stories when the need arises. For this to happen, the product owner should continually refine the product backlog with the help of the development team. In this progressive refinement and re-prioritization of work items, also known as *backlog grooming*, the larger, vague work items are split into smaller and more detailed work items. This is illustrated in Figure 7.3.

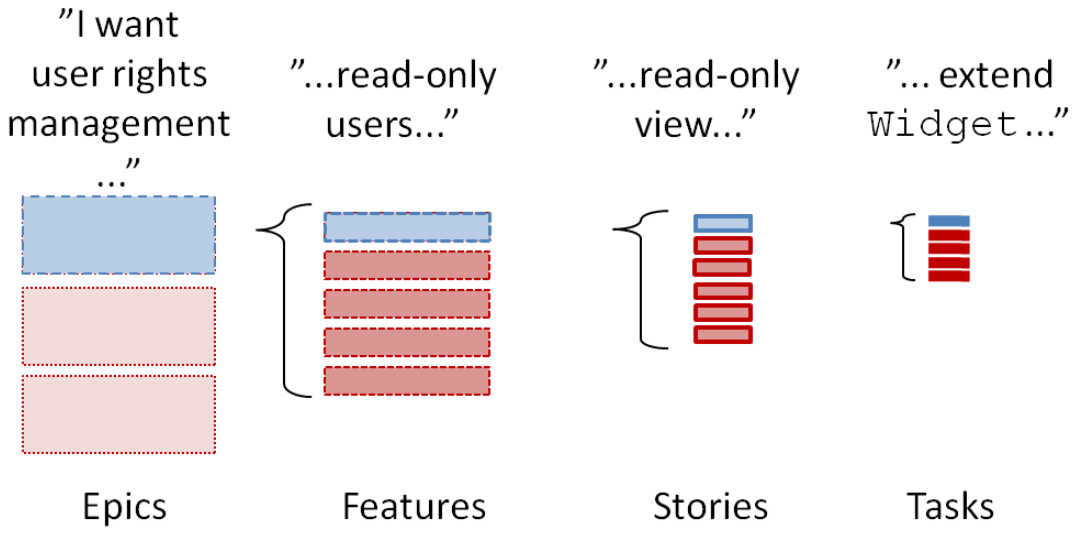


Figure 7.3: Breaking epics to features, stories and tasks (Leffingwell's model[18])

Indeed, during the last few years, frameworks for referring to requirements on different abstraction levels have started to emerge in the literature. Table 7.1 displays the proposed frameworks we are so far aware of, organized according to

[18] Figure 7.3 has been adapted from scalingsoftwareagility.com

the planning horizons of the Cycles of Control framework explained in Chapter 3 (Figure 3.2).

Table 7.1: Requirement abstraction levels by different authors

Cycles of Control	(Gorschek & Wohlin 2006)	(Vähäniitty & Rautiainen 2008)	(Shalloway, Beaver & Trott 2009)	(Galen 2009)	(Vlaanderen et al. 2009)	(Leffingwell forthcoming 2011)
Business mgmt	-	Vision & business goals	Vision / initiative	-	Vision	Strategic product theme
Product & service mgmt (multiple releases)	Product goal	Epic	Business capability	Vision & mission	Theme	Epic
Release (project) mgmt	Feature	Iteration goal	Feature	Epic	Concept	Feature
Iteration mgmt	Compo-nent	Backlog item	Story	Story	Requirement definition	Story
Heartbeat	Function	Task	Task	Task	-	Task

The frameworks in Table 7.1 originate from sources that discuss how agile development can be scaled beyond its original context of an individual, relatively independent team. However, reflecting the terms to the planning horizons of the Cycles of Control framework and Figure 7.2, we see that they can also be used to express long-term product and release plans. For example, using the framework from Leffingwell (forthcoming 2011), a company's strategy for achieving its business goals would be reflected in the strategic product themes and their investment levels. The roadmap for a particular strategic theme would be expressed in terms of epics and features, the planned contents of the current release would be fleshed out on the feature level, and the ongoing iteration in terms of stories and tasks. Note that despite of the terms used, all of the work items on different levels can and even should be thought of as user stories of different granularity (Leffingwell forthcoming 2011, Pichler 2010).

From the perspective of using the frameworks in Table 7.1 for expressing long-term plans and monitoring the progress of ongoing development against such plans, it is problematic that the vast majority of the practitioner books or research articles on agile software development do not take a clear stand on whether it is important to retain the trace of how the smaller work items were split from the larger ones. This is particularly well exemplified by the following excerpt:

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 as noted earlier in this section. 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 2010, 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. Another example of the vagueness surrounding the topic can be found from a systematic review on strategic release planning models (Stober & Hansmann 2010, pp. 77-78). While the authors dedicate two pages to discussing the importance of having a system of hierarchical goals from the overarching company strategy down to the actual coding work, they do not directly relate this to requirements management or the product backlog per se.

We take the position that in the light of what little research (Lehto 2010, Lehto & Rautiainen 2009) and opinions (Shalloway, Beaver & Trott 2009, Leffingwell forthcoming 2011, Ktata & Levesque 2009, Savolainen, Kuusela & Vilavaara 2010, Leffingwell 2007) there are on the subject, the trace of how the smaller, 'child' user stories have been split from the higher level, 'parent' user stories is crucial to retain.

Without the trace of how the individual iteration-level work items contribute to higher level objectives, monitoring progress of development in terms of these higher level objectives becomes very difficult. The missing feedback loop beyond the level of iteration-level work items often is, in fact, the missing link between agile software development and product roadmapping and release planning, in other words, product management. Also, knowing the higher level objectives and the business context may be useful for providing guidance for the developers' decision-making regarding the implementation details as well (Lago, Muccini & Vliet 2009).

Section 7.3.3 below further discusses the issue of traceability, the need for goal-oriented requirements structures and explains the ATMAN framework for linking strategy with action (Figure 7.4).

7.3.3 From strategy to action and back again

This section describes the ATMAN framework for linking long-term product and release plans with agile software development. The essentials of the framework are illustrated in Figure 7.4 and discussed below. The terminology and concepts of the framework have been related to the levels presented in Dean Leffingwell's

agile requirements model (Leffingwell forthcoming 2011), as it seems currently to be the most prominent framework related to the subject[19].

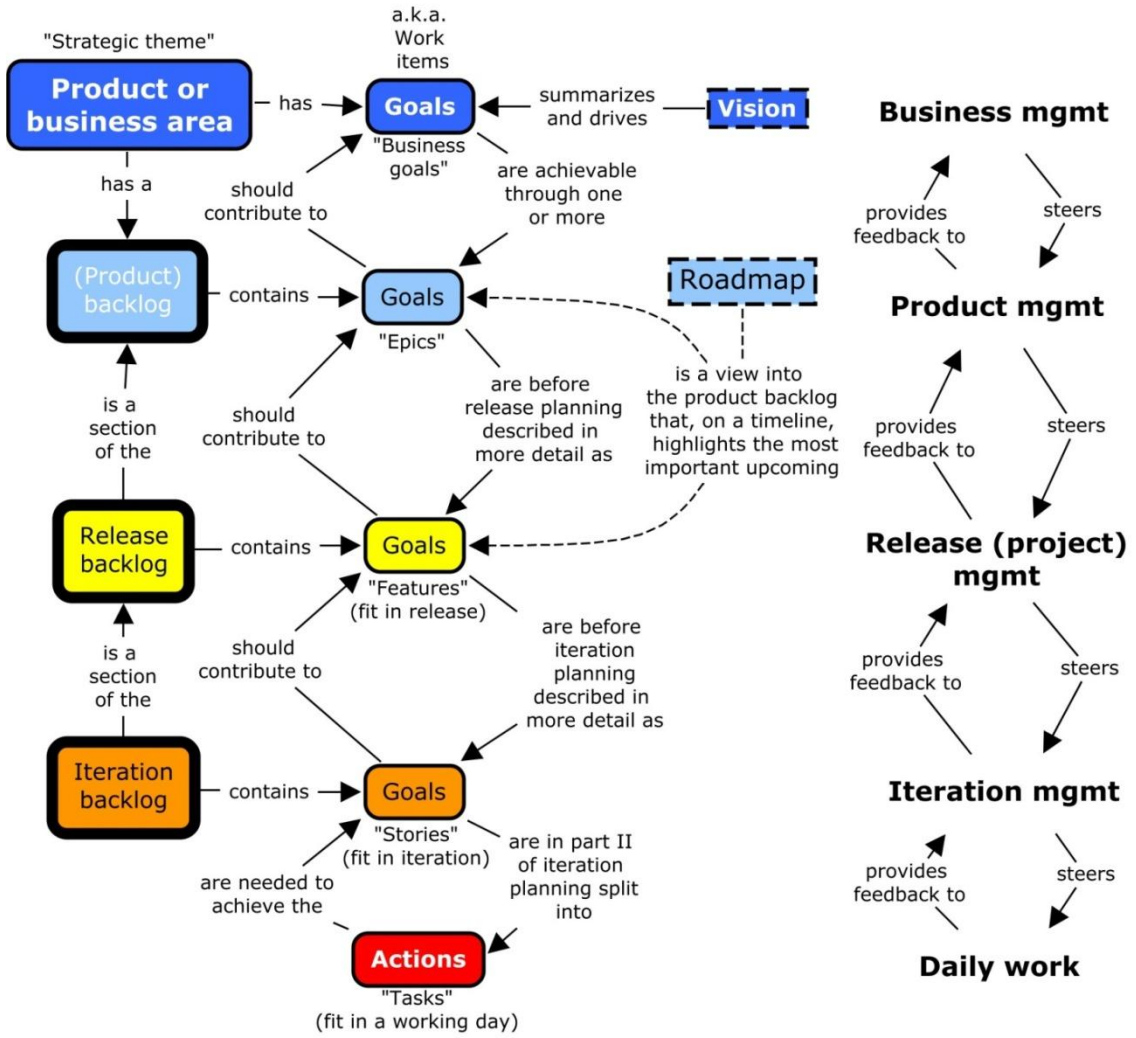


Figure 7.4: The ATMAN framework for linking daily work with product and business goals[20]

Figure 7.4 depicts how business level goals and the vision for the product (or business area) can be linked with the developers' daily work via a system of hierarchical goals. These 'goals' are commonly referred to as backlog or work items. Goals that are 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.

[19] The most notable influences on the ATMAN framework since Vähäniitty & Rautiainen (2008) are Lehto & Rautiainen (2009), Lehto (2010) and Leffingwell (forthcoming 2011). The first two further developed the framework from Vähäniitty (2008) to more explicitly describe a hierarchy of backlog items. The third coined Epics, Features and Stories as currently the most widespread terminology for referring to a three-level requirements hierarchy in the context of agile software development.

[20] The objects with a dashed line denote concepts that have not at the time of writing been implemented in Agilefant as concepts of their own.

In line with the definitions provided in Section 7.3.1 (*Roadmapping, release planning and the product backlog*), a Roadmap is a view into the product backlog that highlights the most important Features and Epics that are planned to be pursued in the foreseeable future.

The high-level goals expressed in long-term plans (e.g. business goals and epics) are, via the processes of roadmapping and release planning eventually split into short-term objectives (e.g. features and stories), which in turn get expressed as actionable tasks via iteration planning. When the trace of how the 'smaller' goals result from the 'larger' goals is kept, the result is a hierarchical system of goals per ongoing development activity. The 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 splitting a larger goal into smaller ones may, especially as development proceeds, yield new information that results in modifications to the original high-level goal.

In Chapter 8 and Chapter 9 we further explain this from the perspective of portfolio management and how it should be understood in agile software development.

Chapter 8: Portfolio Management and Agile Software Development

Jarno Vähäniitty

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 products and/or business areas. This is commonly referred to as product portfolio management. 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. And, in daily work, the development people choose which task(s) from which activities get attended to next.

In this chapter we explain how these levels of portfolio management manifest themselves in the context of agile software development (Section 8.1). Then we present a series of steps on how to set up agile portfolio management, coupled with examples of each step based on experiences from case companies (Section 8.2) and two alternative approaches to setting up agile portfolio management found from literature (Section 8.2.10).

8.1 Levels of portfolio management in an agile enterprise

There is always more than one potential initiative that requires attention from the same resource pool, and thus, effective portfolio management is always crucial – whether explicit or not. However, in the context of agile software development, it becomes clear that instead of a singular portfolio management process, there are actually multiple levels of portfolio decision-making, that

each should connect both with each other as well as with product management decision-making of the corresponding level.

There are *levels of portfolio decision-making* that we as well as several other authors have found relevant in managing an agile enterprise. These levels are 1) *Setting investment levels for business areas*, 2) *Setting product and business goals*, 3) *Development portfolio resourcing*, 4) *Resolving mid-iteration emergencies*, *and* 5) *Time management conducted by individuals.*

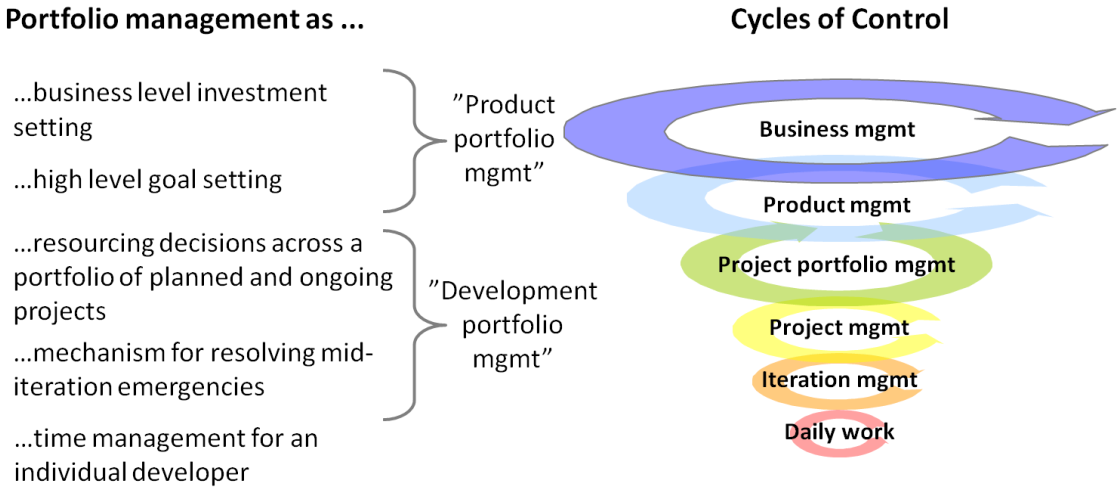


Figure 8.1: Levels of portfolio management in agile software development

As illustrated in Figure 8.1, the two first mentioned levels match quite well with the concept of *product portfolio management*, while the rest cover the decision-making we referred to as *development portfolio management* back in Chapter 2. The levels of portfolio management are further explained in Sections 8.1.1-8.1.5 below.

8.1.1 Setting investment levels for business areas

At the top, portfolio management in an agile enterprise is about setting investment levels for product and/or business areas of the company. For example, Dean Leffingwell's framework for agile requirements (Leffingwell & Aalto 2009) and enterprise agility (Leffingwell 2009) discusses portfolio management as an enterprise's top-level activity for defining *strategic product themes* (or *investment themes*, or simply *themes* for short) and their investment levels.

Leffingwell's themes differ from work items such as epics or stories in that their priorities are not expressed in rank-order, but rather as percentage-based investment levels. Figure 8.2 provides an 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.

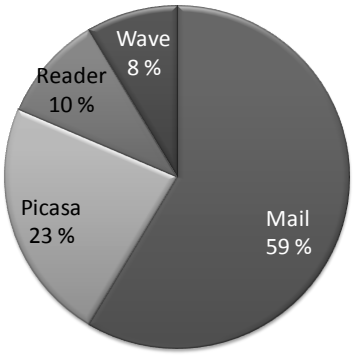


Figure 8.2: Resource allocation according to strategic product themes as an example

For a focused organization, only a few themes should be active at any one time (Leffingwell & Aalto 2009). While a work item that has a low priority may never be worked on, the top-ranking work items within a 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.

This view of portfolio management is congruent with the view of portfolio management adopted in the literature on software product management (see Chapter 3: The Gap in the Literature).

8.1.2 Setting product and business goals

Portfolio management also takes place in the form of updating the product vision and setting high level goals for the business and product areas so that they are compatible with the current investment level. For each *product/business area* (or *strategic product theme*, as Leffingwell calls them), there should be a *product/business vision statement* and the most important *business goals* should be spelled out (see Figure 7.4 on p. 124 and Figure 8.3 below).

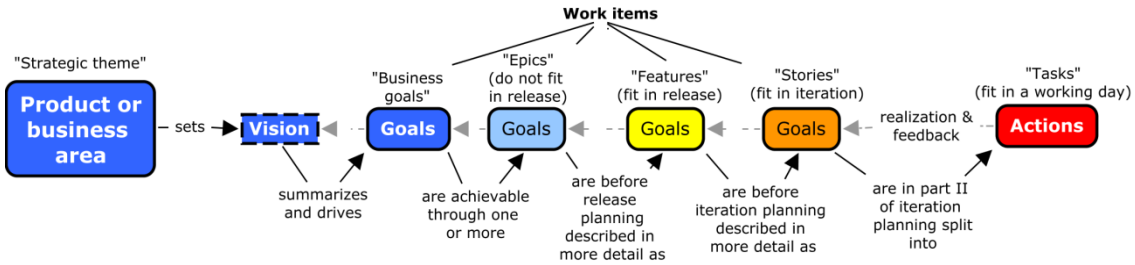


Figure 8.3: From investment levels to product/business area vision, goals, actions – and back again

In Leffingwell's framework (Leffingwell & Aalto 2009) portfolio management on this level deals with the prioritization of *epics*. These are derived from themes, and are the highest level expression of a customer need. Leffingwell defines Epics simply as stories which are estimated as "too big to be realized in a single release". In Leffingwell's framework, the instantiation of themes occurs first through epics, then through features and finally through stories (see Figure 7.3 on p. 121). Our framework (Figure 7.4 on p. 124 and Figure 8.3 above) distin-

guishes between business goals and product epics, but otherwise the frameworks are compatible.

8.1.3 Development portfolio resourcing

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 projects*.

For example, Shalloway's concept of *lean portfolio management* (Shalloway, Beaver & Trott 2009) entails 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 Leffingwell's use of the term.

Figure 8.4 illustrates Shalloway's approach of how the development resources are allocated to develop the most important business features.

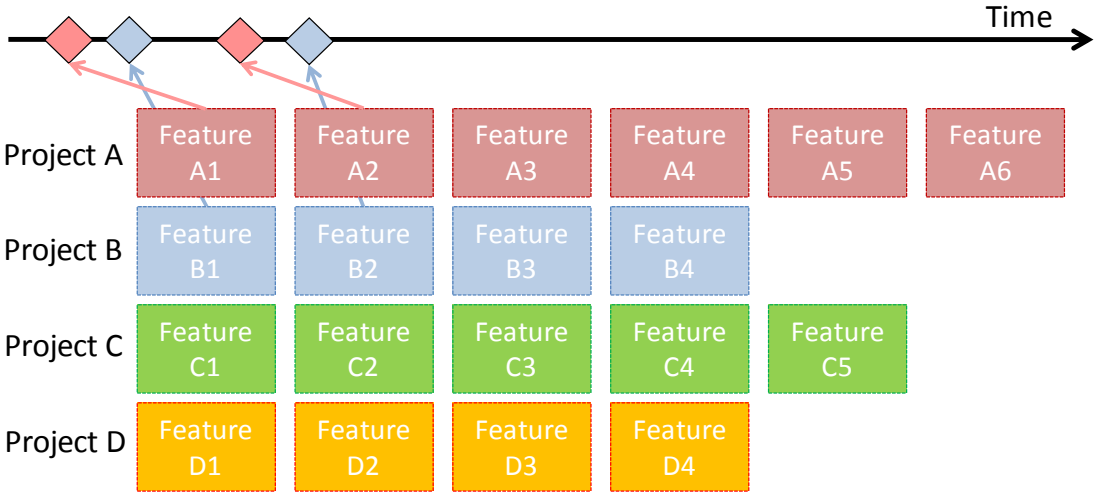


Figure 8.4: Portfolio management as decision-making on short-term project resource allocation (Shalloway, Beaver & Trott 2009)

In the first iteration, the most important business features 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. Thus, in terms of Leffingwell's framework (Leffingwell forthcoming 2011), 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 in (Shalloway, Beaver & Trott 2009). Another voice in favor of this kind of 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 tends to lead to local optimization. Instead, portfolio management can be more effectively 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 in a list called the "project portfolio backlog". Decisions about which projects will continue, be 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. 310)

The approaches discussed above seem to assume that all of the ongoing activities that require the developers' attention follow an agile life cycle, have up-to-date backlogs, and have synchronized cadences. Indeed, a 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 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. 3)

While this is plausible, **in most organizations, all, or even most of the development resources' activities may not be run using an agile life cycle – or even conducted as distinct projects**. This is not discussed in the literature on agile software development beyond warning against such situations, as stated by Krebs (2008):

Don't mix agile and non-agile projects in one portfolio. (p. 137)

Also, even with respect to those activities that are managed using an agile life cycle and have up-to-date product backlogs, it may not in practice be easy to prioritize their contents against each other on a sprint-by-sprint basis (Hodgkins & Hohmann 2007). The higher level context for the small work items may not be evident because the trace of how the smaller work items were split from higher level goals is missing (Lehto 2010, Lehto & Rautiainen 2009). For more

on the issue of work item traceability, refer back to Chapter 7 (especially Sections 7.3.2 and 7.3.3).

8.1.4 Resolving mid-iteration emergencies

At the iteration level, portfolio management is responsible for resolving mid-iteration emergencies that require escalation. Literature on agile software development is generally against the notion of making mid-iteration changes to iterations' staffing or contents. However, because of concerns regarding revenue or the customer satisfaction of important clients, 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 with the crisis even if this compromises the completion of what was being worked on (Rothman 2007). In these situations, stripping activities of resources and/or putting them on hold in order to salvage something of more importance is a portfolio management decision.

8.1.5 Time management conducted by individuals

In general, literature on agile software development 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, more often than not, this is far from the case in reality (Rothman 2007, Rothman 2009, Haapala 2010). Thus, in the less-than-optimal but all-too-common-situation of people having multiple responsibilities, the bottom-up time management of individual workers can be seen as one final level of portfolio management. 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 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 proposes that in addition to "project work", there is also *periodic work*, *ongoing work* and *emergency work* (Rothman 2009). 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 should be transformed into periodic work whenever possible**, which is something we advocate as well. For example, checking and responding to email is something that most people can most of the time restrict to a pace of two times per day.

However, there is no evidence either way whether including ongoing and periodic work in the team's sprint backlog is an optimal way to communicate the possible unsustainable situation of too many concurrent duties to management. Also, current literature on agile software development so far seems to provide little guidance as to how the transition of a team or an individual from multitasking between several development efforts to a single-backlog situation should be carried out in practice.

We recommend that **instead of forcing people to be assigned to a single activity only, your tooling should help enlist, collect and communicate the duties and the respective workload that an individual may have (and most likely has) from his multiple concurrent assignments**. This also means that your tooling should somehow support the concepts of periodic and ongoing work as described above.

We also recommend that you do not 'abandon' time management as something that either happens or does not, and cannot be helped. While making most of their time **is** everyone's personal responsibility, you should make sure that people know that it actually is their responsibility, as well as collect good time-management practices (such as the Pomodoro Technique[21]) and spread knowledge about these across your company.

8.1.6 Summary: portfolio management decisions on different levels

To complement the single product / business area view presented in Figure 7.4 (*The ATMAN framework for linking daily work with product and business goals* on p. 124) and Figure 8.3 (*From investment levels to product/business area vision, goals, actions – and back again* on p. 128), Figure 8.5 below provides a more detailed illustration of the function of portfolio management in moderating the flow from strategy to action and back and the basic "parties" involved.

[21] http://www.pomodorotechnique.com/

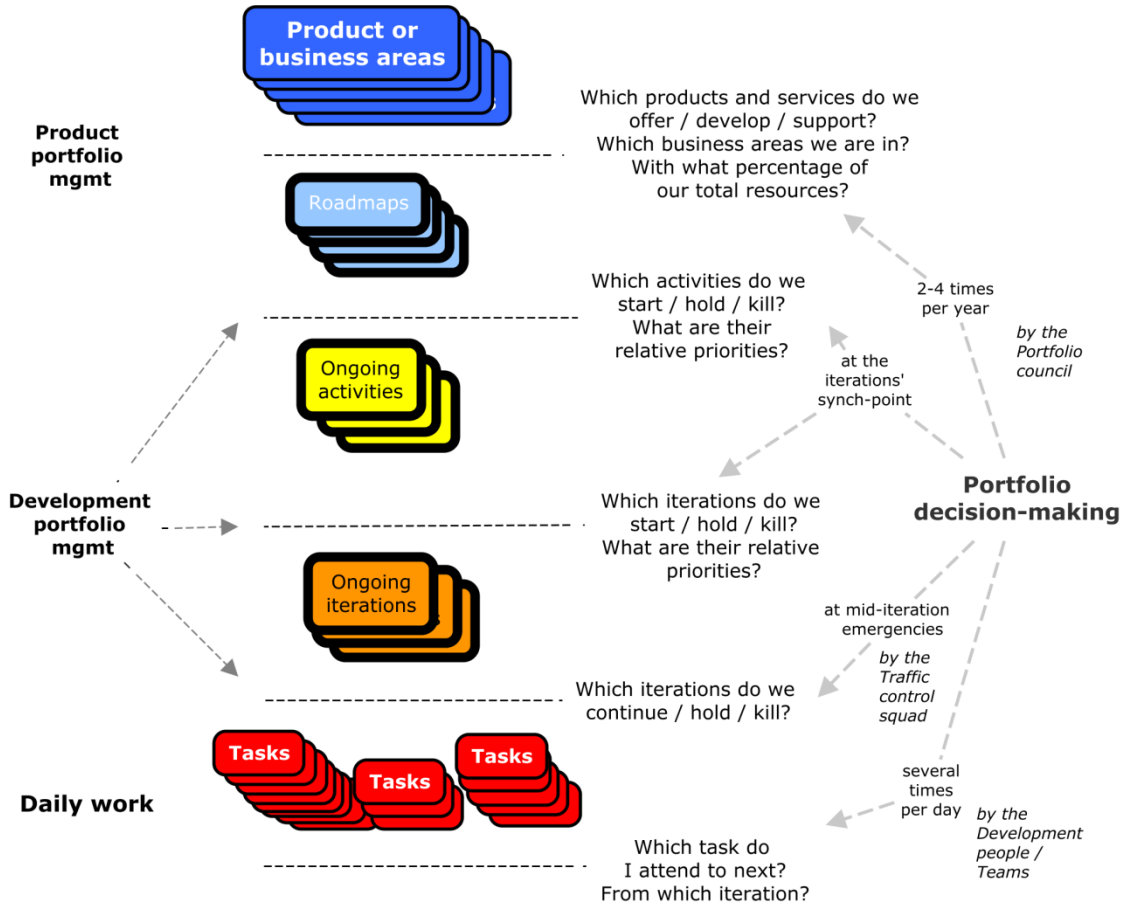


Figure 8.5: A map of portfolio decision-making on different time horizons

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 set of products and/or business areas. This roughly corresponds to the notion of deciding on strategic business themes and the percentage of total corporate resources to spend per theme as defined in (Leffingwell forthcoming 2011)

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 (see Figure 7.4) is responsible for prioritizing and preparing the backlog of a certain activity, development portfolio management decides according to the situation at hand which of the activities actually get resourced, and what their relative priorities currently are.

Note that in Figure 8.5 *ongoing activities* are meant to include everything that takes up time and attention from the development people / teams, for example customer-specific development, consulting, possible non-project work and so on[22]. In many of the case companies we have worked with, especially non-project work seemed to take up an amount of effort from development people

[22] See the notion of *types of development activity* in Section 8.2.3.

that upon a closer look was both considerable as well as a surprise for both the developers and the managers involved.

8.2 Setting up agile-compatible portfolio management

In our experience, the key steps in setting up and performing agile-compatible portfolio management are:

1. Mapping who is/are responsible for decisions on the various levels
2. Building a publicly visible list of all ongoing activities that require time from development, including the information on who are assigned to which activities
3. Identifying the different types of development activities
4. Setting target spending levels per development activity type that reflect the organization's strategy, and possibly tracking the actual spending
5. Ensuring that incentive systems do not steer people towards local optimization
6. Synchronizing the portfolio
7. 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
8. Agreeing on how decisions affecting more than one ongoing activity are made in urgent, 'emergency'-type situations
9. Curbing excessive multi-tasking by explicitly setting limits to the number of concurrent activities a person can be involved in
10. Keeping the enterprise cadence going!

These steps are further explained in Sections 8.2.1-8.2.9 below.

8.2.1 Step 1: Appoint who is responsible for what

Different decisions belong on different levels (see Figure 8.5), and different roles should participate in making them.

The three "basic parties" responsible for portfolio management decision-making depicted in Figure 8.5 are the *Portfolio council*, the *Traffic control squad*, and the *Development people* (or *Teams* in the context of "pure agile"). The *portfolio council* is composed of the people responsible for the business success of the company, as well as 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*. It 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.

Table 8.1 below features an example of how HardSoft has organized its development decision-making, closely resembling the basic schema described above. The **key roles** on each level are **written in bold**, and the *assisting roles* are written in *italic*. When things are "going smoothly", participation from the assisting roles is not needed. However, when it seems that one or more goals set (for e.g. the iteration, the release, etc.) in the above cycle are in jeopardy, the assisting roles need to participate.

As can be seen from Table 8.1, the key roles that should participate in development portfolio management at HardSoft are product manager(s), the portfolio manager, head of product development (should such exist as a separate role) and the project manager(s).

Product managers represent Business and they are, when necessary, responsible for inviting Business unit managers and/or Sales to participate in portfolio management decision-making. Likewise, it may be necessary to have more experienced developer(s) present to participate in the discussion. These are typically invited by the product or project manager(s).

Depending on the size and complexity of the portfolio (and the degree of tool support), it may be a good idea to dedicate a person solely (or mostly) to portfolio management. In most of the companies we have seen, it has been difficult to get the development management process up and running without a person who can set aside the time to take sufficient action. Whatever the case, the portfolio manager should possess sufficient presence and charisma to keep the meetings in line. Also, to ensure the neutrality of the role, the person acting as the portfolio manager should not have direct product, project, or business unit management responsibilities.

Note that the terms *product manager* and *product owner* should be extended to include anyone who "owns" a crucial development activity (or a development activity type). For example, if customer service (or maintenance, deployment projects, installation, training, etc.) are essentially powered by the development people, the managers of the respective functions should take part in portfolio management.

Table 8.1: Key decisions on each 'cycle' at HardSoft and the roles involved

Cycle	Decisions	Roles involved
Corporate governance	Overall direction and area of operation for the company in terms of its business units, their interaction, and investment levels, attitude towards growth	*Board* **CEO** **Business unit managers** **"Department" heads** (sales, development, products)
Business unit management	Business goals and revenue, product vision, release cycles, identifying the types of development activities	**Business unit manager** **Sales** **Product managers** **Head of development**
Product management	Product roadmaps, resource requirements, release goals for e.g. individual segments / solution offerings / technology	*Business unit manager* *Sales* **Product manager** *Product owner* *(Lead) developer*
Activity[23] portfolio management	Balancing the goals and the overall resource demands set in roadmaps through launching (killing / freezing) development activities, monitoring spending levels, prioritizing the development portfolio, setting criteria for selecting and prioritizing development activities and conducting periodical evaluations, identifying dependencies between ongoing activities, setting and enforcing portfolio control points	*Business unit manager* *Sales* **Product managers** **Portfolio manager** **Head of development** **Project managers** *(Lead) developers*
Activity management	Features, Stories, release-level prioritization	**Product manager** **Product owner** *Team*
Iteration management	Stories, tasks, implementation order, iteration level prioritization, …	*Product manager and/or Sales* **Product owner** **Team**
Heartbeats / daily work	Personal backlog item and task lists, updating Effort left –estimates, …	*Product owner* **Team**

8.2.2 Step 2: Compile a list of all ongoing activities

According to Rothman (2009) as well as our experiences, it is rather 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, a first step to take is to list all ongoing activities that require time from at least the development people – whether they are conducted as explicit projects

[23] "Activity" is anything that can take up time from development, e.g., a release project, customer support, etc. See Section 3.3.3 Development portfolio management.

or not – and marking on the list the people who are involved in each activity. You might also want to list those activities that are on hold or immediately upcoming.

This list should be made publicly available (preferably so that it is on a central place where people can easily see it as they go about their duties at the office) as well as kept up-to-date on a continuous basis (see Section 8.2.7 *Steps 7-8: Define enterprise cadence via portfolio control points* for details).

Figure 13.6: Portfolio overview in Agilefant 2.0.4 on page 202 depicts an example of a list of planned and ongoing activities, along with the relative priorities of the activities, the people involved in each activity, as well as the status of each activity as deemed by the person responsible for the activity in question.

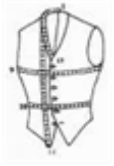
8.2.3 Step 3: Identify development activity types

Identifying the different *types of development activities* the developers are attending to creates a framework and terminology for thinking about and discussing the performed activities as an explicit portfolio (Wheelwright & Clark 1992). Adding structure to the portfolio by identifying the development activity types of the company helps in seeing how each planned or ongoing activity contributes to the big picture, and makes it easier for management to decide what mix is currently appropriate and why (Vähäniitty & Rautiainen 2005).

One of the common problems is underestimating the time that is being spent on all other activities besides the defined development projects. Because of the high degree of resource sharing – at least in small companies – the development portfolio should include all of the activities that require attention from the developers, whether or not these activities actually involve “product development” in the strict sense.

Table 8.2 below displays an example the types of development activity identified at HardSoft, along with their *targeted and actual spending levels* (see explanation in Section 8.2.4 below)

Table 8.2: Types of development activities, their target spending levels (TGS) and actuals (Act.) at HardSoft

Symbol	Activity type	Includes	TGS	Act.
	Customer-specific product development	Customer projects (includes design meetings and meetings for planning security issues - neither were previously not taken into account when planning schedules) Delivery and production testing The finalization phase (previously left out)	40%	35%
	R&D	Product development projects (all 'larger stuff' is projected) Prototyping, hacking-for-the-sake-of-interest, design meetings	25%	5%
	User support and maintenance	Planned (reviews, servicing runs, small tweaks to delivered systems' configuration, updates and their preparation) Unplanned i.e. must reserve time for these (on-call alerts and resolving them, resolving 'red flags')	15%	25%
	Training and consultation	Customer training Lecturing	5%	5%
	Sales support	Sales work done by development people Trade fairs & preparing for them Bidding & preparing bids for contracts Small development tasks requested for sales & promotional purposes	5%	15%
	Self-improvement	Internally driven (e.g. writing team level action guides, adopting the document management system, personnel training, improving development processes and tool support, taking part in university research projects) Externally driven (e.g. certification)	5%	?
	Administrative tasks	Business unit management meetings Team meetings Logging spent effort	5%	?

8.2.4 Step 4: Set target spending levels

Establishing target spending levels means deciding how much in relative terms should in an ideal case be spent on each activity type. The resulting balance should reflect the acceptable risk level and the strategy of the company (McGrath 2000). Once target spending levels have been set, the work items in each activity type can be prioritized against the level set.

See Table 8.2 above for example targeted and actual spending levels at HardSoft Inc. The strategy set by management will require an increase in the offerings' level of productization, which in turn requires concentrating more on R&D. Looking at the distribution of the actual effort spent in Table 8.2, it seems that the most likely paths to correct this are to find ways to streamline *User support and maintenance* and cut back on developers' involvement in *sales support*.

As another example, Figure 8.6 below displays the types of development activities identified at three software companies. Figure 8.6 also displays the target spending levels for the company Odysseus Inc.

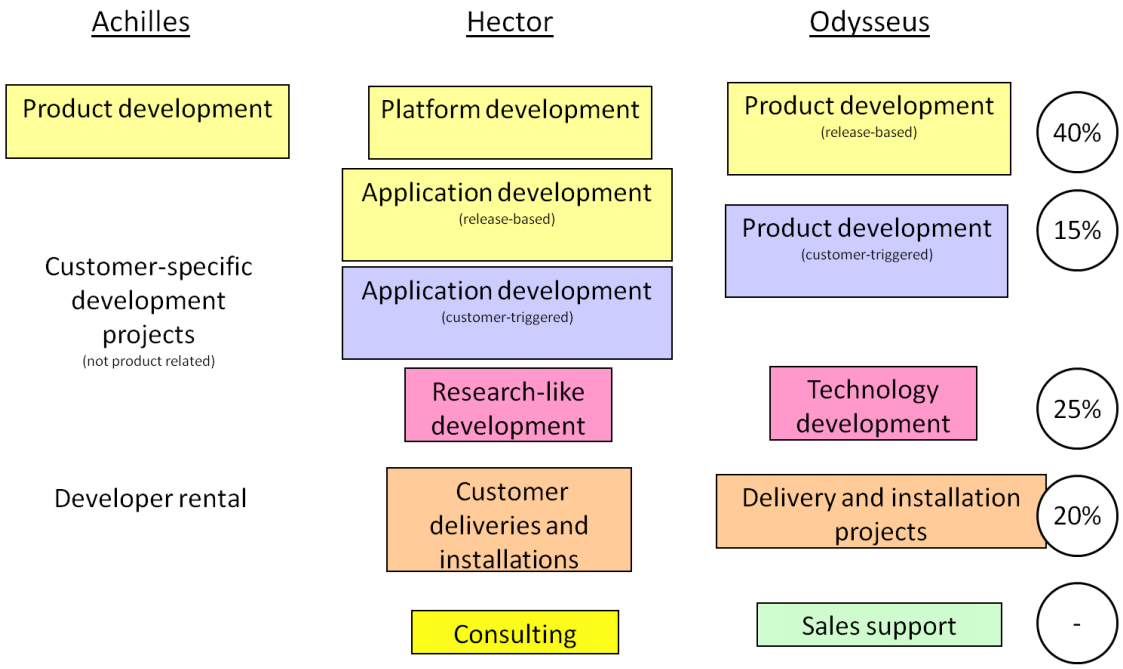


Figure 8.6: Types of development activities identified at three companies

Modern development methodologies tend to consider tracking spent effort in hours as either directly unproductive (McCarthy & McCarthy 2002) or at best, uninteresting (Schwaber & Beedle 2002). Nevertheless, in real-life companies there may be valid reasons for tracking actual hours spent for billing and/or accounting purposes[24]. Surprisingly, however, the need to reflect target spending levels against the actual hours spent is not necessarily one of these valid reasons! If you track the estimates for backlog items' (or tasks, requirements, user stories etc.) remaining effort, you can get a reasonably accurate idea of how your actual spending is matching your targeted spending even without tracking hour spent per se. This is achieved by comparing the realized velocities of your completed past iterations – grouped by development activity type – against the set target spending levels.

8.2.5 Step 5: Check incentive systems

By the time the previous steps have been completed, it has probably become painfully obvious if you have dysfunctional incentive systems in place. That is, incentive systems that steer people to behavior that causes local optimization. Refer to Section 4.2.5 for more on this issue.

8.2.6 Step 6: Synchronize the portfolio

Besides the fact that different development activities compete for the developers' attention, these development activities should in many cases also be managed differently. For example, the process for developing a new major release is likely to have different emphasis than the process for conducting customer-specific tailoring, not to mention making customer deliveries or training the customers. In our earlier work (Rautiainen et al. 2006) we have identified *cadence*[25] and the resulting *control points* as the backbone for managing software development. Cadence supports persistence and forces convergence while retaining the flexibility to change plans and adapt to changes at specific time intervals in the control points.

[24] See: http://danube.com/blog/michaeljames/tracking_hours_spent_appropriate_and_inappropriate_usage

[25] Actually, the term used by us in the past has been *rhythm* instead of cadence. However, most authors nowadays use the term cadence – and it is, at least in English, a more accurate term for the phenomenon in question. Thus, we are also using the term *cadence* in this book.

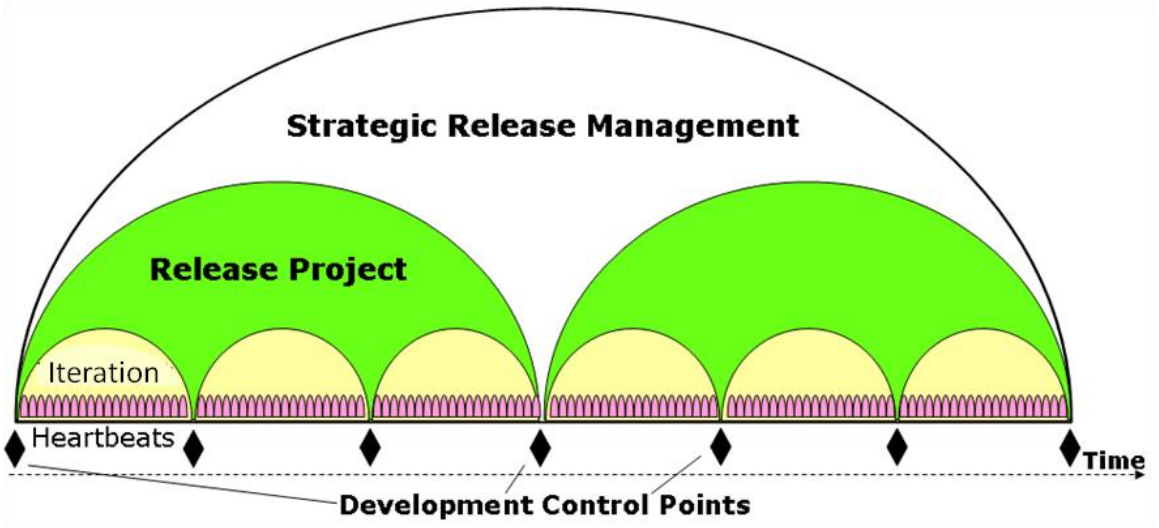


Figure 8.7: Cadence and control points in a release-based, incremental development process

Figure 8.7 shows an example cadence for release-based software product development. The time horizon long-term product and release planning, referred to in Figure 8.7 as "*strategic release management*", spans two *release projects*, each release is built in three iterations and the daily work is coordinated and synchronized with suitable practices in *heartbeats*. Each time horizon begins and ends in a control point to plan it or wrap it up, respectively. Specific agendas of control points should vary depending, e.g., on the time horizon and the development activity type (Rautiainen 2004).

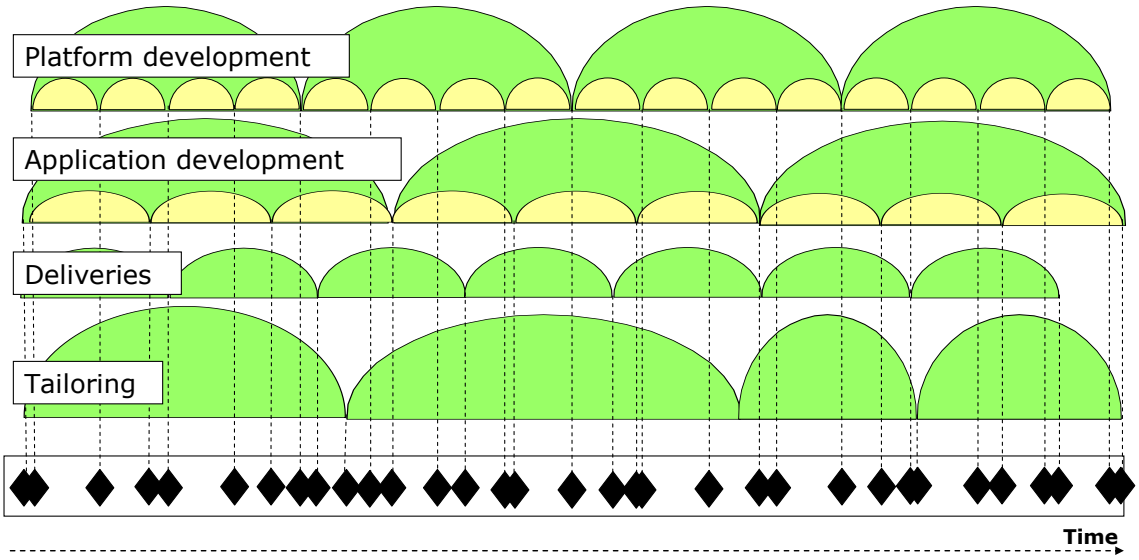


Figure 8.8: An out-of-sync development portfolio

Figure 8.7 displays the simplified case of the cadence for a single type of activity (release-based development). However, in Figure 8.8 above we have illustrated what the cadence for an entire development portfolio, consisting of four types of development activities, might look like. In Figure 8.8 each development activity has its own cadence. Setting a suitable cadence for a development activity en-

tails understanding what kind of cadence suits the 'customers' of the activity in question. For example, events and the rhythm of the market directs when product releases should be made. However, the nature of the offering (for example, the time needed for testing), the internal capabilities of the company (for example development process effectiveness and personnel skills) as well as how much effort can be spent considering all the other tasks at hand constrain what is possible.

Setting a development cadence creates control points that (with the exception of the heartbeat level) may require the attention of portfolio level decision-making. Thus, as illustrated in Figure 8.8, there is a danger that those responsible for portfolio decisions on various levels become overloaded with requests for attention due to the sheer volume of control points required by even the four ongoing development activities shown in Figure 8.8.

An out-of-sync portfolio[26] leads to problems in resource planning and allocation. When push comes to shove, the types of development activities with close customer involvement tend to override release-based product development, which in turn makes longer term planning of release-based product development both difficult and frustrating.

Portfolio synchronization means organizing the control points for different development activities so that the overall enterprise cadence becomes as simple as possible. For this to succeed, the cadence of all types of development activities should be similar. For example, if the longest iteration time horizon for any development activity is 4 weeks, the other development activities should have iteration time horizons that are 1, 2, or 4 weeks. In this way the entire enterprise is synchronized at least every 4 weeks, which can be used as the time interval for portfolio control points (see Section 8.2.7 *Steps 7-8: Define enterprise cadence via portfolio control points*). Even if conducting customer deliveries and doing consulting would not by themselves require control points on the iteration level, they should adhere to some kind of cadence for the benefit of resource planning and allocation for the entire portfolio.

The notion of portfolio synchronization is also supported by virtually all of the existing (though scarce) grey literature that discusses portfolio management in the context of agile software development. Synchronizing the iterations makes it feasible to commit the resources for a fixed period. Provided that the organization's cadence or "enterprise iteration" is short enough, this helps alleviate a fire-fighting mentality as cross-project trade-offs are possible to make proactively and on a more continuous basis.

[26] Many, or perhaps most companies are actually doing multi-tasking on a non-synchronized portfolio that has an irregular cadence; more on this in Chapter 9.

8.2.7 Steps 7-8: Define enterprise cadence via portfolio control points

Ultimately, the cadence of an enterprise is set via portfolio control points. We have identified three basic kinds of portfolio control points: *roadmap updates*, *portfolio reviews* and *traffic control meetings*. These are illustrated in Figure 8.9 and further explained below.

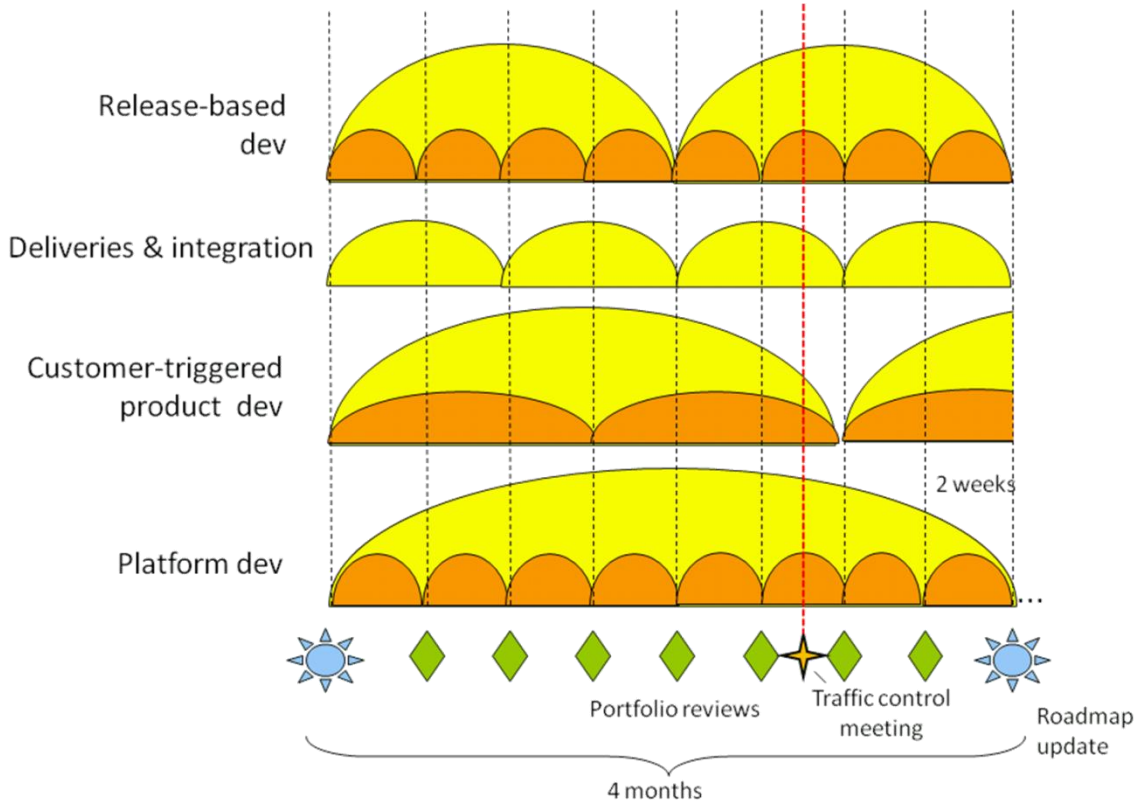


Figure 8.9: A synchronized development portfolio with three types of control points

Portfolio Reviews set the enterprise cadence, and they are the primary mechanism for ensuring that the ongoing activities are aligned with strategy. Portfolio Reviews look at the ongoing development activities as defined by their internal control points, but keep the entire portfolio in mind when dedicating resources and setting the scope for the upcoming set of development iterations. The objective is basically to freeze the resources and scope for the upcoming set of iterations. If the portfolio reviews are held often enough, firefighting is minimized as cross-project tradeoffs are made proactively on a continuous basis (Harris & McKay 1996). Portfolio Reviews require the attention of both Business and Development, but depending on the exact roles and responsibilities, the representation of Business need not be as extensive as in Roadmap Revisions (see Section 8.2.1 *Step 1: Appoint who is responsible for what* on page 134). For portfolio reviews to work, portfolio synchronization is in most cases necessary.

Table 8.3 below describes the responsibilities (from the perspective of portfolio management) of the major roles involved in portfolio management before, during and after portfolio review meetings.

Table 8.3: Roles and responsibilities related to Portfolio reviews

Role	Responsibilities in preparation of, during, and after the meeting		
Business	Help the product manager(s) as needed to prepare the business case	Final word on resource spending per ongoing activity and the relative priorities until the next portfolio review; sanity check the decisions against company strategy and financial goals	Support people to act according to the decisions made
Product manager(s) & Product owners (includes the heads of development resource –dependent functions , e.g. customer service)	Prioritize your product backlog(s) to include enough stuff in line with the currently approved roadmaps to work on until the next portfolio review (in case you got all of the resources you wanted); Figure out which resources you'd want; prepare the business case(s) to match your needs	Presenting what the approved roadmap(s) have in store for the upcoming period (e.g. in terms of stories and features), what these require in terms of resources, and what can be achieved if the requested resources are granted	Check early & often that the work done indeed is such that it fulfills the related stories and features; keep activity status up-to-date
Portfolio management	Make the portfolio management visualization and the dashboard up-to-date with current status. Gather information about actual progress. Organize the meeting (i.e. invite the participants, make sure they are coming)	Act as meeting chair; keep a (suitable) memo of the discussion and decisions made; present actual spending since the last portfolio review (if available); expose unplanned work since the last portfolio review; sanity check the decisions against historical performance	Update the portfolio visualization; monitor whether the decisions made in the meeting are being acted out
Project manager(s) / Product owner(s)	Get up-to-date on how the upcoming iteration goals are doing and possible impediments	Inform others of how the currently ongoing iterations are doing and why; strive to remove the impediments	Blow the whistle on unplanned work; let portfolio and product management know of impediments; keep activity status up-to-date and make sure work items' status reflects their real state
Development	Help the product manager(s) by estimating efforts as needed for prioritizing the product backlog(s); help project manager(s) to understand current status; tell portfolio management if something unplanned has been demanding your attention; reserve time for joining the meeting when asked to contribute to the discussion	Aid in the discussion and decision-making when you can make a contribution	Focus on the daily work; inform product owner/project manager of impediments; inform product owner / product management of unforeseen technical difficulties; keep stories and tasks up-to-date

Roadmap updates deal with issues such as product visions and release strategies, and should involve a procedure for long-term planning, such as product roadmapping (Kappel 2001). Roadmap updates require the attention of people from both Business and Development.

Table 8.4 describes the responsibilities (limited to the perspective of portfolio management) of the major roles involved in portfolio management before, during and after roadmap updates. While the Roadmap update is referred to here as 'a meeting', it might as well take the form of a two-day strategy retreat, complete with evening program, etc. Also, at least in small companies, it is a good idea to involve the entire staff in the process.

Table 8.4: Roles and responsibilities related to Roadmap updates

Role	Responsibilities in preparation of, during, and after the meeting		
Business	Prepare for revising the target spending levels and the criteria for selecting and prioritizing development activities; check whether the identified types of development activities still match what's actually going on. Organize the meeting (i.e. invite the participants, make sure they are coming)	Act as meeting chair. Set a balanced resource allocation that matches the company strategy for the competing set of proposed releases and other development activities. Keep a (suitable) memo of the discussion and decisions made.	Support people to act according to the decisions made; check when possible that the work progresses towards the set business goals; participate in the portfolio mgmt process (see Table 8.3 above)
Product manager(s) & product owners (includes the heads of development resource –dependent functions , e.g. customer service)	Do a preliminary revision of the product roadmap(s) representing your own view of where the product should go.	Present the revised product roadmap(s) and justify the resource demands for the upcoming release(s)	Keep your backlog(s) groomed; participate in the portfolio management process
Portfolio management	Analyze the past period in terms of target vs. actual spending levels, strategic alignment, financial value, conformance to the decisions made in the prev. roadmap update and the portfolio reviews, and the controllability of the development portfolio	Present a summary of the past period, sanity check the decisions made against historical performance, request adjustments to the portfolio management process as necessary	Participate in the portfolio mgmt process
Project manager(s) / Product owners	Help portfolio and product management where needed; reserve the time to participate to the meeting	Sanity check the decisions made against historical performance	Participate in the portfolio mgmt process
Development	Help portfolio and product management where needed	Sanity check the decisions made against historical performance	Participate in the portfolio mgmt process

For more on roadmap revisions, see e.g. (Vähäniitty 2004).

Traffic control meetings are essentially event-triggered Portfolio Reviews. Business realities may make an absolute adherence to the principle of freezing resource allocation until the next portfolio review impossible. Defining and allowing Traffic control meetings increases the likelihood of systematic and conscious decision-making when mid-iteration changes have to be made. In addition to making the needed changes in priorities and resourcing, a Traffic control meeting should also analyze and record the root cause that led to the situation. This makes it easier to spot similar situations in advance, as well as provides a baseline for estimating how often Portfolio reviews are likely to be needed, which in turn promotes realism in planning. When a Traffic control meeting is needed, it may or may not be necessary to call the entire team responsible for portfolio decision-making. While the number of people that need to be involved depends on the size of the "traffic jam", the deciding factor is whether the smaller group of people is able to solve the problem and be accountable for the trade-offs made.

The responsibilities of a Traffic control meeting are otherwise identical to that of a Portfolio review meeting (see Table 8.3 above), with the addition that the portfolio manager is, with the help of others, responsible for analyzing and recording the cause(s) of the traffic jam.

8.2.8 Step 9: Curb excessive multitasking

While agile software development literature generally advises strongly or very strongly against individuals or teams multi-tasking on several activities, we deem this challenging in practice. This issue is explored more in-depth in Chapter 9.

However, while an absolute adherence to single-team/individual-attends-to-a-single-activity-only may be impossible, it is quite easy to go overboard with multi-tasking (and many, if not most companies are doing this already). Thus, at this step you should look at whether organization-wide restrictions on the number of activities a person or a team can be involved in should be in place, and if Work-in-Progress should be limited (for this Kanban could be an alternative, see Chapter 12)

8.2.9 Step 10: Keep the enterprise cadence going!

Getting people to adhere to the enterprise cadence set up by the portfolio control points and sticking to it can be challenging. Ideally, the portfolio council is able to set and keep the resource allocation fixed for the duration of the organization's enterprise iteration. 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. We believe that most organizations

indeed are such; at least, most of the case organizations we've worked with were. Thus, it is quite plausible that such a situation is fairly common.

In our approach, the fact that mid-iteration conflicts eventually occur was addressed by the *traffic control squad*, a forum in which mid-iteration conflicts between activities are resolved in the light of the prevailing 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. Having a nominated traffic control squad increases the chances for systematic and conscious portfolio decision-making to take place when mid-iteration conflicts occur, and helps keep the enterprise cadence going.

8.2.10 Setting up agile portfolio management in the literature

Below, we present two approaches for setting up agile-compatible portfolio management as presented by (Rothman 2009) and (Poppendieck & Poppendieck 2009) that both, when properly carried out, deal with many of the steps presented in Sections 8.2.1-8.2.9. In terms of published research, there is little evidence or suggestions related to agile-compatible portfolio management. Two industrial experience reports published in the Agile 2007 conference were found using IEEEXplore (Hodgkins & Hohmann 2007, Tengshe & Noble 2007). These approaches resemble the one described by Rothman below.

According to Rothman (2009), the first step in setting up portfolio management 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 prioritized against each other in a rank-ordered list, and the result of this ranking should be published along with an explanation for the 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. The evaluation and ranking of projects should then be revisited at iteration boundaries, and preferably, the iterations across different activities should be synchronized.

In Poppendieck's 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 is made of how much of the total capacity the activity should be allowed to expend (see Table 8.5).

Table 8.5: 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 capacity

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 based on what currently seems to be the most valuable initiative for the category in question.

Chapter 9: Agile Development Portfolio Management

Jarno Vähäniitty & Ville Heikkilä

Most advocates of agile software development as well as the majority of the literature advice against assigning a single individual or team to work on multiple development activities such as release projects. However, we believe that working on multiple activities is something that in practice can only very seldom be completely avoided, and trying to avoid it completely only leads to multi-tasking "under the radar", so to speak. So, as your typical managed-by-fire-fighting organization is transforming to a more disciplined agile way of working, simultaneous work on multiple projects and teams should to some degree be allowed – as long as the resulting situation is visible. This chapter starts with explaining why a team or an individual in practice almost always has multiple activities to attend to (Section 9.1). Then we explain what working concurrently on multiple development activities means from the perspective of backlog management in order to ease your company's transformation into a more agile way of working without strictly enforcing the one-team-one-activity limit (Section 9.2). Last, we explain how the degree of working on multiple simultaneous development activities and the related planning overhead can be brought to a more acceptable level (Section 9.3).

9.1 Why have teams work concurrently on multiple projects?

Like explained in Section 8.1 (*Levels of portfolio management in an agile enterprise*, page 126), portfolio decision-making happens – or at least, should happen on multiple levels. We also explained how portfolio management manifests itself during iterations and ultimately in developers' daily work (see Section 8.1.5 *Time management conducted by individuals* on page 131).

When you have one team that works on only one iterative activity at a time, such as a product development project, the team members choose which work item

they will next attend to. This decision-making is quite simple, since a developer has only a single sprint backlog to pick work items from. This is also the basic Scrum model at its simplest, and with it there are numerous benefits such as no inter-team dependencies, there is no switching from one project to another during an iteration, and progress monitoring is simple.

However, what goes on in real-world organizations seldom resembles this. Instead, developers in practice have several different activities that compete for their attention, for example, working on the next release of a product, responding to customer support requests, and preparing for conducting a training session at a customer. Or, a person with specialized skills may belong to two teams, which have been assigned to two different projects which have their own distinct sets of work items to get done during their ongoing iterations, which may or may not have the same start and end dates.

A development project can benefit greatly from the simplicity of the basic Scrum model of limiting multiple assignments to the maximum of one. This is because the comparison is made to the hardly optimal *everybody-doing-too many-things-concurrently-and-prioritization-occurs-by-fire-fighting* situation.

However, there are downsides to enforcing teams and individuals being assigned to a single activity. First, the rest of the activities tend to suffer, as they are still managed with the old prioritization-by-fire-fighting mentality, but are crippled in the sense that certain people can no longer be used in "saving the day" when needed. This can actually be a serious hurdle from the perspective of agile adoption – especially in small companies – as the majority of people actually end up suffering as the result of providing one team the peace and quiet needed for success!

Second, even if all activities are conducted using the basic Scrum model, the single-activity-only mode has several limits – which are, to a degree, recognized and even addressed in some Scrum trainings nowadays[27]. For example, if for a business reason you need to show progress on two activities - say, a product development project and a customer-specific customization project at the same time, this can't be handled by the single-team-single-project model. Or, if in the name of customer satisfaction support requests are to be responded to during an ongoing development iteration, this should be accounted for in the iteration planning.

[27] For example, at least on those two-day product owner courses we have attended; the advice given is to reserve in iteration planning the time for the additional activities that are expected to be handled during the iteration. However, at least on those courses we've attended, this is not discussed in-depth. In this chapter, we intend to take the discussion further, and show the implications of the advice.

9.2 Controlled multi-tasking with floating backlogs

To support teams and individuals who for one reason or another have to work on multiple concurrent development activities to properly manage their backlog and adhere to agile/lean principles, you need what we call *floating backlogs*.

The backlog is 'floating', because it is not related to any specific product/business area, or even a time horizon. Instead, it refers to **a prioritized list of stories merged from the product/release backlogs of multiple product/business areas that a particular team (or individual) is responsible for at a particular moment in time**. The priorities of the work items as well as the intended resource spending per activity have been negotiated and agreed upon by the product owners responsible for the product/business areas in question. This is illustrated in Figure 9.1 below.

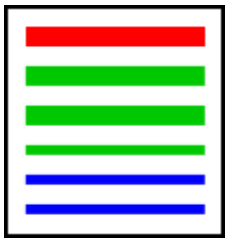


Figure 9.1: Work items from several backlogs merged into a single, prioritized backlog for a team

A team should not pull more stories from a product/release backlog into its floating backlog than it thinks it can accomplish within a certain cadence taking into account all the other activities the team members are involved in. The team should reserve time for those activities according to the agreement of the respective product owners.

Figure 9.4 (*Development portfolio management for two teams working on three concurrent* activities) on page 153 shows an example of how backlog items from multiple activities are pulled into the teams' backlogs. Next, we shall explain the example illustrated in its entirety in Figure 9.4 on page 153 one step at a time.

In our example there are two teams, Team A and Team B, which are attending to three activities: a project that develops the next release of a product (hereafter, *Development* for short), another project that is customizing a version of the product for an existing customer (hereafter, *Customization* for short), and the continuous activity of responding to customer support requests (hereafter, *Support* for short). All of the activities have their own backlogs and "product owners" who are responsible for prioritizing and grooming the backlogs. In our example, Development is color coded as red, Customization as blue, and Support as green. Work items are represented by colored rectangles, with the size of the rectangle representing the amount of effort estimated as needed to get the work

item done. Support does not have a cadence in the same sense as Development or Customization. Support items are removed from the support backlog as they get done. New work items are added to the support backlog whenever support requests arrive, and the customer support manager grooms the support backlog on a daily basis. This example covers two Development and three Customization iterations. The horizontal axis represents a timeline and the half-circles denote the iterations. Colored diamonds (e.g. ◆) denote a session for planning the contents of the next iteration for the activity in question (red for Development, blue for Customization, and green for Support). The backlog of the activity is drawn inside each half-circle. This is illustrated in Figure 9.2 below.

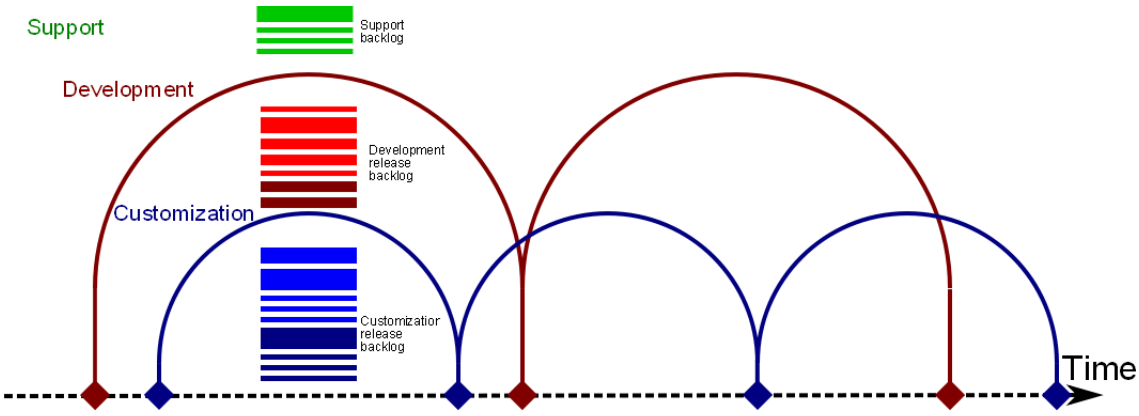


Figure 9.2: The cadence of the ongoing activities

As said earlier, there are two teams that both are available to take on work from the three activities (see Figure 9.3 below).

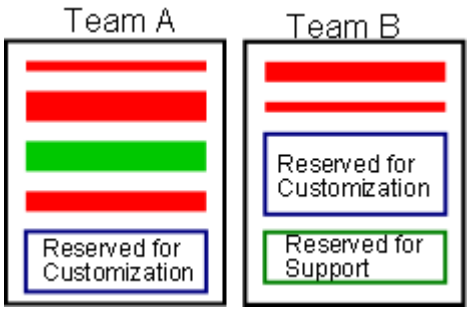


Figure 9.3: Teams A and B and their floating backlogs at the first planning session

In Figure 9.4 (page 153), the backlogs of both teams at each planning session are shown on a timeline. The hollow-tipped arrows (—▷) going into the backlog items denote a work item being pulled into a team's floating backlog. Lines ending in a dot (—●) denote a work item getting done. Arrows from a work item to another depict the movement of work items inside and between the teams' floating backlogs over time.

Let's now walk through the rest of the example step by step. For this, we recommend that you print a copy of the entire illustration (Figure 9.4 on the following page) or open it from **http://tinyurl.com/floatingbacklogs** so that it can be viewed together with reading the paragraphs that follow.

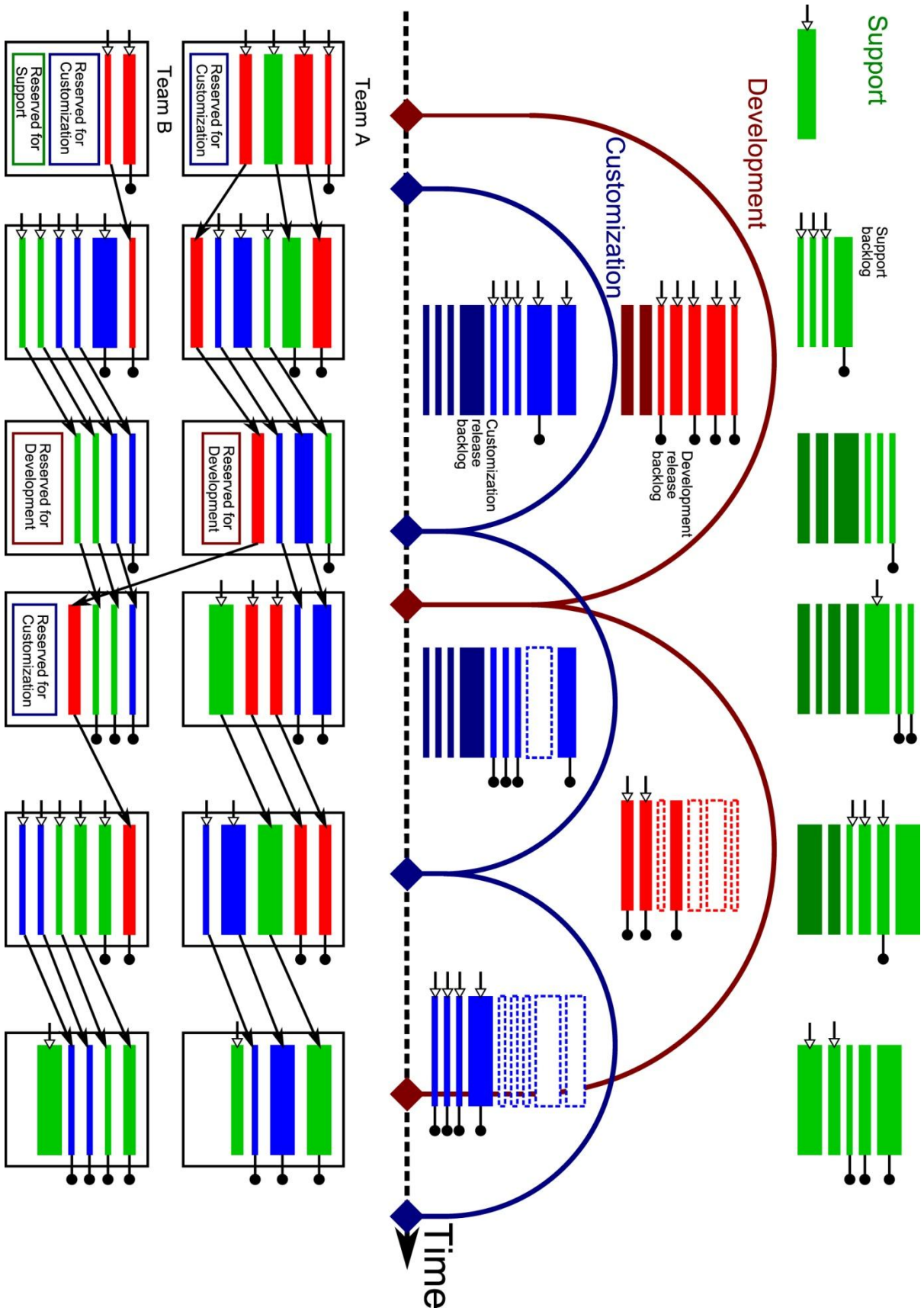


Figure 9.4: Development portfolio management for two teams working on three concurrent activities

At the first Development planning, Team A pulls three Development work items from the Development backlog, and the single available Support item from the Support backlog. The Development, Customization and Support product owners have agreed that a certain amount of the Team A's total available time is reserved for the upcoming Customization iteration 1, which has been planned to start during Development iteration 1. Team B pulls two Development items, with the rest of its time reserved for dealing with the soon starting Customization iteration and those Support requests that might arrive during the first Development iteration.

By the time Customization iteration 1 starts, Team A has completed one development item. Team B has also completed one Development item. In addition, three support requests have arrived and the respective Support items have been created and prioritized by the customer support manager. When planning Customization iteration 1, Team A and B pull two and three Customization items respectively. Team A also pulls one and Team B two support items in their backlogs.

Because of the shorter cadence of Customization, Customization iteration 1 ends before Development iteration 1. By this time, Team A has done one more Development item as well as its most important Support item. Also, Team B has completed its most important Development items as well as one Customization item. By this time, three additional support requests have arrived and been prioritized. In planning Customization iteration 2, neither of the teams pulls new items but instead, time is reserved for pulling Development items in the soon upcoming planning of Development iteration 2.

By the planning of Development iteration 2, Team A has completed one Support item, and Team B has completed one customization item. Two new support requests have arrived, and the respective Support items have been created and prioritized. When planning for Development iteration 2, one Development item that has not yet been started is agreed to be transferred from Team A to Team B. In addition, Team A pulls two Development items and one Support item. Team B doesn't pull any new items, but reserves some time for the upcoming planning of Customization iteration 3.

By the planning of Customization iteration 3, Team A has completed one, and Team B has completed two Customization items. In addition, Team B has completed two Support items. Two new support requests have also arrived, with the respective Support items added and prioritized in the Support backlog. When planning Customization iteration 3, Team A and Team B both pull two new Customization items. In addition Team B pulls two Support items.

By the end of Development iteration 2, both teams have completed all their Development items, and the remaining Support items are pulled. By the end of Customization iteration 3, both teams have completed the Customization and Support items they had in their backlogs. The two Support items pulled at the end of Development iteration 3 are not yet done, and work on them continues.

9.3 Towards a feasible level of multiple concurrent assignments

The model described above may seem quite complex. However, **it is quite close to how many non-agile companies are actually trying to work.** However, there are several important differences that separate it from your typical uncontrolled *everybody-working-on-multiple-things-ad-hoc-mgmt-by-fire-fighting* mode.

For starters, there are true cross-functional teams that hopefully have already 'jelled', and the majority of the ongoing activities have defined cadences. Second, the development activities that compete for the teams' attention have owners who maintain prioritized backlogs. Third, the owners of the competing activities are able to agree to a common set of priorities for the activities themselves as well as target spending levels – at least for the duration of the activity with the most frequent planning cadence. Thus, instead of committing to sets of backlog items, the teams actually commit to putting in a set amount of effort per activity during the planning events. The job of the product owners of the activities is to match the work items they wish to get done to the available level of effort. Collaboration in setting the investment levels per activity is crucial, because it is in the mathematical sense impossible to determine what the 'next most important item' to attend to is. For example, which would be more important: to attend to the tasks needed to accomplish story #2 of the most important project, or the tasks needed to accomplish story #1 of the project that has currently been ranked as second in importance for the ongoing set of iterations? Or, if something needs to be added to the most important activity, should something be removed from it, or should the scope down be made to a less important activity? In our model, these decisions are made in accordance with the investment levels agreed upon in the planning events. Finally, the teams and product owners are reacting to how the work actually progresses. In the example, during the first Development iteration, Team A was not able to complete a development item they had pulled, and subsequently the item was transferred to Team B at the next planning event.

At this point, you may still be thinking that the approach described above is too complex to work in practice. And you are right: as the number of activities and teams increases, the time spent in the planning events increases, especially as both the teams as well as all of the activities' owners must be present at each

event. The teams must also take into account any dependencies between the items, which becomes harder and harder as the number of activities and teams increases. Unnecessary context switching may also happen, since different types of activities are mixed in each team's backlog. All of these are the natural downsides of applying agile outside of its traditional sweet spot of a single team working with a single product owner and a single backlog.

However, the most important unnecessary complexity in the above model is that of not having the activities' planning cadences synchronized. In the kind of situation described, the planning cadences of those activities that actually have a defined cadence should be synchronized[28]. This reduces the effort spent on planning, as the concurrent activities can be addressed as a whole making it easier for the product owners to negotiate and agree on the investment levels until the next planning event.

When feasible, you can take the simplification even further and reduce switching costs by assigning each team to only one activity per iteration. In this case, teams can still be switched between iterations.

[28] See *Step 6: Synchronize the portfolio on page* 35.

Chapter 10: The Agile Requirements Refinery

Kevin Vlaanderen, Slinger Jansen, Sjaak Brinkkemper & Erik Jaspers

Due to the complexity of software products, with a large variety of stakeholders, long lists of requirements and a rapidly changing environment, Software Product Management (SPM) is a complex task. Relatively little scientific work has been performed in this area. Especially regarding agile SPM, little work exists. It seems that the complexities of SPM have been left lurking behind the Product Owner (in Scrum) or resident expert user (in eXtreme Programming) role. One case study describing the use of agile requirements engineering is described in (Pichler, Rumetshofer & Wahler 2006). However, the paper does not provide details regarding the agile requirements engineering process. Part III of this book shows one way to link long-term product planning and agile development. Greer and Ruhe (2004) elaborate on agile release planning by providing an iterative optimization method. Collaboration between product managers and development teams in challenging environments, such as where no complete requirements are available, is investigated in (Fricker et al. 2010). In a comparative case study by Fogelström et al. (2010), a misalignment was identified between the agile principles and the needs of pre-project activities in market-driven development. They state that the differences between agile methods and the needs of market-driven software development may threaten product development by disabling effective product management.

In this chapter, we describe in which way software product management can be performed in a Scrum development context. We explain an agile SPM method based on Scrum, which improves the ability to handle large amounts of complex requirements in an agile environment. We also provide a set of

useful lessons learned that aid in the implementation of Scrum-inspired SPM alongside agile software development. This chapter is based on a case study that has been reported in two articles (Vlaanderen et al. 2009, Vlaanderen et al. 2011).

10.1 An Approach to Agile Software Product Management

10.1.1 Background: Scrum development method

The Scrum development method was proposed in 1995 by Ken Schwaber (1995), at a time when it became clear to most professionals that the development of software was not something that could be planned, estimated and completed successfully using the common 'heavy' methods. The Scrum method is based on the work of Pittman (1993) and Booch (1995), and adheres to the principles of agile software development.

Central to Scrum is the idea that many of the processes during development cannot be predicted. It therefore addresses software development in a flexible way, by inspecting and adapting. The only two parts that are fully defined during a software development project are the first and last phase (planning and closure). In between, the final product is developed by several teams in a series of flexible black boxes called 'sprints'. No new requirements can be introduced during these sprints. This ensures that the final product is being developed with a high probability of success, even within a constantly changing environment. This environment, which includes factors such as competition, time and financial pressure, maintains its influence on development until the closure phase.

The backlog is an important instrument in the Scrum process. The following backlogs play a part in Scrum development:

- **Product Backlog (PB):** The PB is central to the Scrum method. The PB contains a prioritized list of all items relevant to a specific product. This list can consist of bugs, customer requested enhancements, competitive product functionality, competitive edge functionality and technology upgrades (Schwaber 1995). Once a requirement has been fully specified, with the approval of a developer, the requirement can be copied from the PB onto the Development Sprint Backlog.
- **Development Sprint Backlog (DSB):** Each team that participates in the software development process maintains its own DSB. All requirements that are assigned to the development team at the beginning of a sprint are put on their DSB. Every requirement is decomposed into several tasks, which are then assigned to specific team-members. The Development Sprint Backlog is fed by the product backlog with items that have been fully specified.

The DSB enables continuous monitoring of the progress of development work, while the PB enables weekly to monthly renegotiations about the priorities for each requirement.

10.1.2 Adapting Scrum to Agile SPM

The development of software by large teams of developers requires a steady flow of elicited product requirements. Without this steady flow of requirements, software vendors run the risk of delaying new software releases and bad code due to badly specified requirements, all resulting in the waste of large amounts of resources. To avoid these problems, a functioning team of product managers is required, that can, cooperatively with the development team, supply approved and well-defined requirements. The agile SPM method applies Scrum to maintain a steady flow of new requirements for the DSB.

Furthermore, agile SPM enables a software vendor to flexibly define requirements according to a pre-defined procedure. The pre-defined procedure forces a software vendor to explicitly manage the lifecycle of a requirement, leading to better-defined requirements. Simultaneously, the process remains agile, i.e., some requirements can be defined and implemented quickly, while others move through their lifecycle at a regular pace.

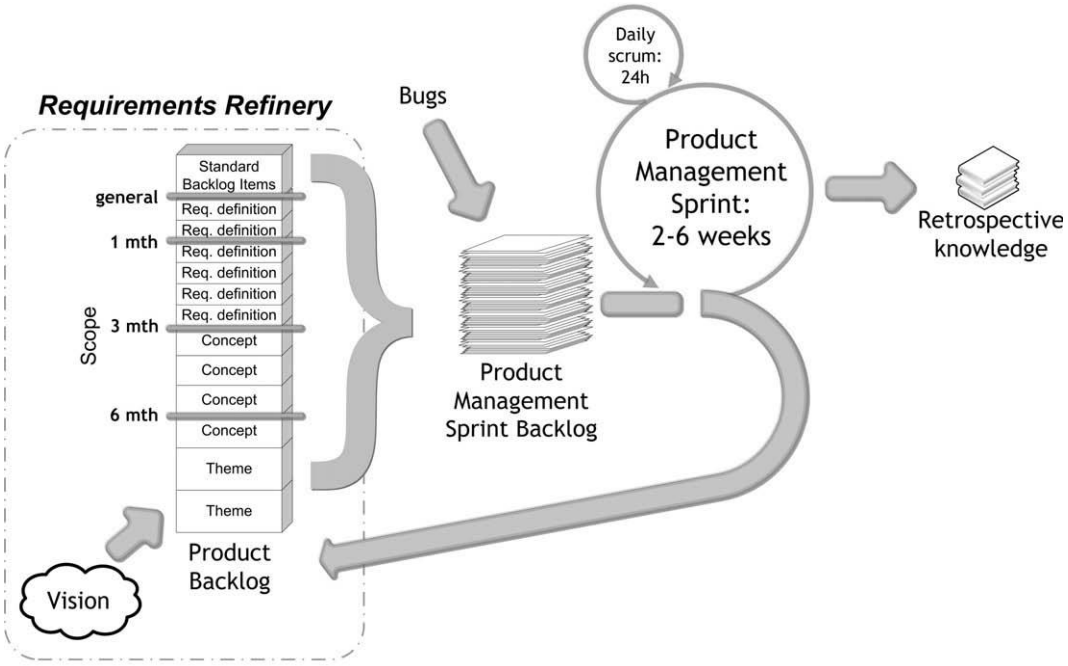


Figure 10.1: Agile SPM Knowledge Flow

Figure 10.1 shows the flow of knowledge within the agile SPM process. The figure is based on the default Scrum development process described in the previous section, and is supplemented with SPM-specific adaptations. In the figure the product management sprint backlog (PMSB) is introduced. The PMSB is an agile SPM concept. It provides product managers with a way of working similar

to that of developers in the Scrum process, using PMSB items to establish division of work, and work planning:

- **Product Management Sprint Backlog (PMSB):** The PMSB contains all items that need to be completed within the sprint by each product manager. The PMSB is fed with items from the product backlog, the full list of themes, concepts, and requirements for a product. The PB feeds the PMSB with items that need further specification before they can enter the DSB.

Scrum and the agile SPM process are similar in the aspects that they both work in sprints, and that both developers and product managers perform tasks according to the shared PB and a team backlog. The main difference is that at the end of a sprint, developers produce a working version of the software, whereas product managers produce refined requirements. Table 10.1 lists the differences between the agile SPM process and Scrum.

Table 10.1: Differences between Scrum development and agile SPM

	PMSB	DSB
Takes work from..	Product Backlog (PB)	Product Backlog (PB)
Demands..	Vision (unspecified requirements), bugs	Specified requirements
Supplies..	Specified requirements	Functional software
Deals with..	Visions, concepts themes, requirements	Bugs, product enhancements, functionality, technology, upgrades, etc.
Works in..	Sprints and daily Scrums	Sprints and daily Scrums
Worked on by..	Product managers	Developers
Puts back onto PB..	Requirements definitions	Finished PB items

The input for the agile SPM process is in most cases an idea or a wish for new functionality, but also new technologies, bugs and upgrades. An idea enters the process in the form of a vision, shown by the cloud at the bottom left of Figure 10.1. During a number of sprints, this vision is then refined several times, going through the agile requirements refinery, which will be discussed in the next section. The main result of this process is a list of further specified themes, concepts, and requirements that can be placed back onto the PB. The requirements

that have been fully specified and approved by a software developer are candidates for the next development sprint.

At the start of each sprint, each SPM team has to prepare its PMSB. Based on the amount of time available and the focus determined by the board, the SPM teams select a set of PB items, such as concepts and visions, and place them on their PMSB. This activity is similar to the sprint preparation as performed by the development teams.

The next step is to proceed with either refining the items that are on the PMSB, or introducing new ideas obtained through customer support, meetings with business consultants, customer sessions, industry analysts and involvement at different types of forums in which market parties are active. During a sprint, each item is refined from its current stage to the next level of detail, i.e. from vision to themes, from theme to concepts, or from concept to requirement definitions.

At the end of each completed SPM sprint, a retrospective evaluation takes place, during which each team looks back at the last sprint, discussing about the aspect that went good or wrong. The results are written down, and from the resulting list, two or three items are chosen to be put on the sprint backlog of the next SPM sprint. This enables the teams to gradually improve the process, learning not only from their own mistakes, but also from those of the other teams.

The agile SPM process also includes bugs from earlier versions. These form an alternative way of generating PB items and do not follow the usual path through the requirements refinery. Instead, they are placed directly on the PB. If the bug can be fixed easily, it goes straight to the DSB. If the bug cannot be fixed easily, however, it will go onto the PMSB, for review and further detailing by the product management team.

Each working day, also known as a Scrum, starts with a Scrum meeting, during which the previous day is discussed. As this session is primarily meant to improve the productivity and the effectiveness of the SPM team, a small set of possible improvements is discussed. This helps avoiding experienced problems in the future. The end-result of an agile SPM sprint consists of the requirements definitions, which can in turn be used by the development teams. The sprint length is equal to the length of development sprints, in order to synchronize the heartbeat of the product management and the development process.

There are three important stakeholder groups in the agile SPM process. First and foremost, the product managers' work process is the one determined by the agile SPM process. Secondly, the product board, consisting of key stakeholders for the product, such as the CEO, the support director, the business consultancy director, the development director, and several representatives from sales departments, determines requirements priority and product vision in a monthly

meeting. Thirdly, the development teams increasingly monitor and approve requirements as they come closer to entering the DSB.

10.1.3 Managing the Backlog: The Requirements Refinery

The structuring of the workflow into sprints and Scrums enables agile SPM to deal with customer wishes. Similar to the Scrum development method, no new items can be added to the PMSB, as it has been finalized at the beginning of the sprint. This means that the SPM team(s) can focus on the work at hand without disruptions. On the other hand, it also requires considerable thought about the structuring of specific tasks, since they need to be completed within the timeframe of one sprint. SPM tasks, however, are not easily restructured into fine grained tasks of up to one month. For this reason, the default Scrum-approach to task management has been substituted by the more fine-grained approach that is described in this section.

This approach, the agile requirements refinery, provides a solution for managing complex requirements. The approach is suited to the characteristics of SPM tasks, and it resembles an industrial refinery in a way that during each sprint or iteration work is being performed on the requirement definitions that appear on the PB, to refine them from coarse-grained to fine-grained. Each refinement, from one stage to the next, can generally be performed within one month or even two weeks. When this is not possible, the item should be split and the resulting items should be placed back on the PB to be picked up again in one of the future sprints. By refining complex requirements according to the abstraction levels of the requirements refinery, structure is added to the backlog that will help in completing the tasks in an effective manner.

Since Scrum itself does not provide guidelines for effectively managing large amounts of requirements of different granularity, a set of stages is introduced. Within the agile requirements refinery, a product functionality vision will generally move through these stages, during which it is refined with details and specifications. The stages are:

- **Vision:** A vision is the starting point for the lifecycle of most requirements. It is an idea, brought up by the company board, a customer or any other stakeholder, and is defined in generic terms. Once the idea reaches a product manager, he or she then converts it into a (set of) theme(s). An example of a vision is the wish to target small enterprises as potential customers for an ERP software package with a light version.
- **Theme:** A theme is the formal elaboration of a vision, describing it in more detail. The product manager defines the envisioned purpose of the new functionality, the business value of the theme, and the involved stakeholders. A theme should briefly describe the business problem from which it originates and the main issues that fall within the theme scope. This can possibly be ex-

tended with a set of provisional requirements. In total, a theme description should not exceed one page of text, in order to maintain clarity. The previously described vision can for instance be translated to the theme ‘small enterprises’, describing its importance and what would be required to accomplish it. In reality, a vision is often so complex that it can be refined into multiple themes. To ensure the technical feasibility of a theme, it is reviewed by the development teams.

- **Concept:** Themes are broken down into smaller pieces called concepts. A concept is a high-level focal point within the theme, consisting of a set of solution stories that can later be used to deduct detailed requirements. The elaboration of each concept results in a document describing product drivers, product constraints and the concept scope. The description should briefly explain the necessity of the concept, while remaining clear and detailed enough to be useful for the definition of detailed requirements. The ‘small enterprise’ theme could for instance be converted to a set of concepts such as ‘productX Lite’, describing the high-level requirements of a product suited to the needs of small enterprises. Each concept definition should be checked with the software architects. Also, the developers help estimate whether the concept is sufficiently defined to further split up the concept into requirements.
- **Requirement definition:** The detailed definition of requirements is performed in three steps, of which only the first one is performed by the SPM team(s). SPM translates the concepts into a list of requirement definitions without going into a lot of detail. Requirement definitions consist up to this point of a description, a rationale and a fit criterion. The latter describes a constraint that must be met in order for this requirement to be successfully implemented. To ensure feasibility and compatibility with other requirements, each requirement definition should be checked with architects, functional designers or lead developers. After the initial high-level requirement definitions have been determined based on the previously defined concepts, the software development teams then elaborate these into requirements containing a detailed description of some desired functionality, described in sufficient detail to work with. To accomplish this, each requirement definition is first processed during a development sprint by a development team, to ensure that they are feasible, consistent and understandable in a general manner. Then a second pass is made, where the development team ensures requirement clarity, so that each requirement is understood by all teammembers. This results in a list with all relevant requirements and their detailed descriptions, including any necessary diagrams, technical specifications or otherwise necessary information that is required for the implementation of the requirement.

Instead of the hierarchy of requirements presented here, you could try other hierarchies that you find better suited to your context. One hierarchy proposed in literature is the one by Leffingwell (2007) with (Strategic theme-) Epic-Feature-Story (-Task). Another view, a user story maturity pipeline, is presented in (Fisher & Bankston 2009).

With smaller topics, the definition of a vision and a theme might not be necessary, in which case the problem can be placed within an existing theme or concept. They are then elaborated without constructing a vision, theme and/or concept, or they are elaborated with the vision, theme and concept constructed afterwards. In other words, the requirements refinery is not restricted to a top-down approach, but can also be used bottom-up. This is similar to the approach by Gorschek and Wohlin (2006), who identified four abstraction levels on which a requirement can be placed, along with both a bottom-up and a top-down path along these levels.

10.1.4 Timing of SPM sprints and Development sprints

An important aspect of the described agile SPM approach lies in the fact that, like software development, the SPM task is performed according to sprints with a fixed length of one to four weeks (varying per company). However, if the SPM sprint would be performed simultaneously with the development sprint, the deliverables from one team would not always be available in time for the other team's new sprint.

Therefore, SPM sprints should not be performed synchronously with the software development sprints. Instead, they should be shifted back half of the development sprint length. This ensures that the PB is always up-to-date and ready for DSB use once the software development sprint starts, reducing the time between the inception of a requirement and its realization in the product. Also, information regarding implementation progress and the accuracy of requirements sizes and descriptions can flow back from the development teams to the SPM teams.

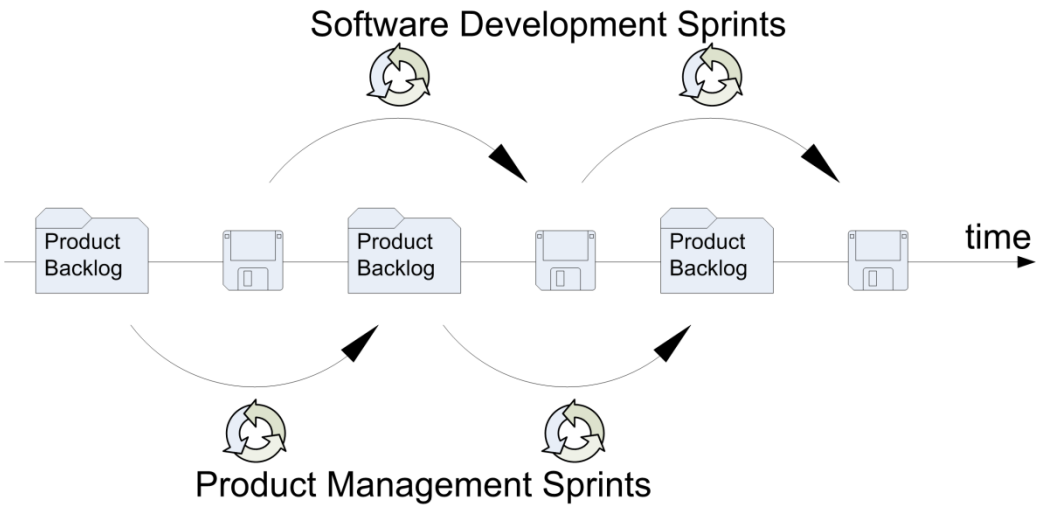


Figure 10.2: Alternating sprints

Figure 10.2 illustrates this concept of alternating sprints. The horizontal timeline shows the synergy between software product management and software development, by switching from a focus on SPM to a focus on software development and back. The SPM team(s) deliver(s) an updated PB while the development teams are developing the next product release candidate (depicted by a floppy). Based on the demo of this release candidate, SPM will then redefine the PB, resulting in continuous double-loop feedback.

Similar to Scrum software development, the PMSB is filled with items from the PB at the beginning of each sprint. The status of completed, canceled or ongoing tasks is continuously kept up-to-date on the PMSB. Each product manager is responsible for keeping the backlog up-to-date as the sprint progresses. Based on the data in the backlog, a burn-down chart is created continuously to allow monitoring of the progress of the sprint.

10.2 Agile SPM in Practice

10.2.1 Standard PMSB items

In order to improve the structure of the PMSB, several standard recurring backlog items have been identified. The standard items, as opposed to incidental tasks, form a basic structure of recurring tasks, mostly with the same amount of hours allocated each sprint. These tasks can be used to create a cadence within the team(s).

Table 10.2 shows an overview of these standard backlog items. On the left-hand side, all standard backlog items related to the SPM sprint are shown. On the right-hand side, all standard backlog items related to the development sprint are shown. All tasks are performed by the SPM team(s).

Table 10.2: Standard backlog items on the PMSB

Spm related backlog items	Development related backlog items
Prepare and attend product board	Backlog preparation
SPM sprint review	Development sprint planning with development teams
Team retro meeting	Development sprint review with development teams
Team allocation overview	How-to-demo stories
Problem and change management	

The following describes the items in Table 10.2:

- **Prepare and attend product board:** The product board consists of several lead positions in the company, such as the CEO and the sales director, who have a major stake in the product itself. Once a month the product management team presents what has been developed and what the future plans are for the product board. The product board contributes in two ways. First, the product management team informs major product stakeholders of the progress of visions and plans that have an impact on the product. Secondly, the product management team is forced to report on their progress, which requires them to evaluate progress speed and SPM process quality.
- **Sprint review:** The sprint review consists of a full review of the SPM sprint. Furthermore, the sprint review leads to an update of the internal and partner information portals of the product. These portals are used to report on the progress of the work and on the upcoming features for partners and sales teams.
- **Team retro meeting:** Once a month during the team retro meeting the internal functioning of the SPM team is discussed. The retro meeting does not specifically address practical problems, but tries to achieve better quality and use feedback to improve the Agile SPM process. The problems are placed as PMSB items and the most important ones are solved or worked on during the next SPM sprint.
- **Team allocation overview:** Throughout the agile SPM process, themes are assigned to teams, consisting of a product manager and a development team. Generally, teams will remain active within that theme. However, when a certain set of requirements that originates from a certain theme can also be implemented by a team that has capacity available, requirements sets might be transferred from one team to another during the team allocation overview.
- **Problem and change management:** The task of problem and change management deals with customer problems and large changes that require

the interference of a product manager. Furthermore, product managers go through the list of reported problems from customers and respond within one month. The response to a reported problem generally consists of declining it, i.e., the problem will not be solved, or accepting it, i.e., the problem will be included in the planning.

- **Backlog preparations:** A basic structure needs to be provided before each sprint, with the appropriate names and task types. This only requires very little work.
- **Sprint planning with development teams:** The sprint planning with development teams consists of an eight-hour meeting. During these meetings product managers and developers negotiate, accept, and approve PB items for the DSB. This is a typical part of the Scrum process.
- **Sprint review with development teams:** During the sprint review the development teams present the functionality they have implemented to the other development teams and the SPM team(s). Developers also defend why the functionality is necessary.
- **How-to-demo stories:** Product managers create how-to-demo stories for the developers who are working within their theme. These how-to-demo stories are specified to indicate to the developers how they should demo the functionality they have implemented during the sprint review. The main reason for the creation of these stories is that developers frequently have a different view of the interesting parts of the functionality they have implemented.

These activities provide an overview of the different standard tasks that are executed in each sprint by the SPM team.

10.2.2 Roles and Tasks

Although the identified recurring backlog items already form useful knowledge within a practical context, the link between the product management tasks and the actual execution of an agile SPM process is still missing. Therefore, a set of roles can be identified that apply to an agile SPM team, each focusing on a specific set of tasks. As fully specialized roles are not common, i.e., a person who handles only one or two kinds of tasks, each role has a characteristic combination of tasks assigned to it.

The first role is the **senior product manager**. This role is mainly management-oriented, reflected by the low amount of time spent on requirements elaboration. Instead, a large share of the senior PM's time is spent on high-level tasks, i.e. on the concept- or theme-level. The remaining time is for a fair amount spent on general tasks, supplier management and other management-

related activities. The senior product manager generally has the biggest influence on issues related to high-level decisions.

The second role is the general **product manager**. A product manager spends a large amount of time on both requirements elaboration and development sprint elaboration. The product manager is responsible for the lion's share of low-level activities related to requirements and concepts. Furthermore, due to the close relation between product managers and the development teams, most of the activities related to the development sprint can be attributed to the product manager.

The third role is the **requirements engineer**, which mainly focuses on low-level work. Almost one half of the time is spent on requirements elaboration. The other half of the time is divided between development sprint preparation, concept-level activities and general activities. The requirements engineer is not an active participant during the standard activities such as sprint review and planning meetings.

10.3 Lessons learned

For companies who are interested in agile SPM, we have derived a set of lessons that should be taken into account when implementing agile SPM alongside an agile software development method.

1. **Alternate sprint cycles for SPM and development:** One of the main lessons learned has been the importance of the alternating sprints. As discussed in Section 10.1.4, the software development and the SPM sprint are both performed continuously, but with a difference in starting date of approximately half of the sprint length. This implies that each SPM sprint ends halfway the software development sprint, ensuring that the PB is ready to be used when the development teams start their new sprint.
2. **Complex requirements are in need of structured detailing:** The essence lies in the division of requirements into themes, concepts and requirements. The structured agile requirements refinery approach has made it possible to effectively manage large sets of requirements of different granularity. Both high-level and low-level requirements are placed on the PB and handled in time by the appropriate person.
3. **Daily Scrum meetings are essential:** The daily stand-ups, or Scrum meetings, that are essential within the Scrum development method, are also valued highly within the agile SPM method. The 15-minute meeting at the start of each day is experienced as a positive, helpful aspect of the process. By providing constructive critique, potential problems can be avoided and existing problems can be solved.

4. **Backlog administration requires discipline:** Strict documentation of all tasks in the PMSB is still difficult to achieve. Although the PMSB can play a useful role in controlling the SPM process and keeping track of the progress of a sprint, the motivation to keep up-to-date the current set of tasks and the amount of time spent on a specific task is still lacking. This undermines the efficiency of methods such as Scrum. However, it should be noted that one of the agile principles is a favoring of individuals and interactions over processes and tools. This means that, as long as the work gets done, project administration becomes less important.
5. **Early collaboration promotes reuse and integration:** Since product managers in a Scrum team cooperatively work on a PMSB and discuss requirements before they have been implemented, reuse and integration opportunities can be spotted at an early stage. We suspect that higher quality software products are built using this approach, rather than using other approaches with less communication during the requirements specification process.

The final three lessons are similar to key aspects of the original Scrum development approach. As the approach described in this section is based on Scrum, this also applies for agile software product management. The first two lessons apply specifically to agile SPM, and we consider them essential to a successful implementation of agile SPM.

Chapter 11: Scaling Up Agile Release Planning

Ville Heikkilä

When an agile development organization grows the basic agile release planning methods become less and less efficient. One way to scale up agile release planning to meet the requirements of multi-team agile development is a method called joint release planning. This chapter first introduces the problem of planning software releases, then describes the joint release planning method and finally motivates the use of the planning method.

11.1 Introduction

Planning the next product release is recognized to be one of the most challenging parts of market-driven product development (Fogelström et al. 2010) and a critical success factor in agile software development projects (Chow & Cao 2008). The main goal of release planning is to find an appropriate scope for a release while taking into account budget, resource, technical, and other constraints (Fogelström et al. 2010, Ngo-The & Ruhe 2008).

Scrum, the most popular agile software development method in 2009 (VersionOne Inc. 2009), was originally created for small-scale software development in small co-located teams (Schwaber & Beedle 2002). The small scale of the software under development is not an integral part of the Scrum process, but employing only a single team creates practical limits for the size of the software when development time is limited. Scrum emphasizes direct and informal communication between team members, which limits the practical size of the development team to approximately eight members (Cockburn 2002). Thus, the only way to scale up the size of the developed software while still holding on to the principle of the direct and informal communication is to employ multiple Scrum teams that simultaneously develop the same software product.

The Scrum process model (Schwaber & Beedle 2002) defines the product owner role, whose responsibility is to manage the development of a product regarding scope and value. The agile release planning process in a single-team single-product development scenario is simple. The team and a product owner discuss the features that could be included in the next release until an agreeable plan is reached. The agreed-upon release plan then acts as a vision for planning the

individual iterations (Schwaber & Beedle 2002, Shalloway, Beaver & Trott 2009, Rothman 2007). During a sprint the backlog items selected to be implemented in the sprint cannot be switched. The rest of the backlog can be re-prioritized by the product owner, but any changes to the release backlog require partial release re-planning (Schwaber & Beedle 2002).

Joint release planning (Heikkilä, Rautiainen & Jansen 2010) is a method for multi-team agile release planning for complex systems. Similar to the single-team agile release planning, the basic idea of the joint release planning method is to gather all development teams and internal stakeholders in a single space to perform release planning together. However, the sheer number of people, requirements and dependencies makes joint release planning difficult to perform efficiently. Scaling agile release planning up to a multi-team environment where many teams are developing the same product at the same time also introduces technical complexity. The teams cannot plan releases in isolation, since requirements are selected from the same product backlog and coordination of who-does-what is required. In an ideal agile world all requirements are independent and can be implemented in isolation by feature teams. In the real world there are often architectural complexities which result in a network of dependencies between requirements (Carlshamre 2002). This also affects the implementation order of the requirements.

11.2 Background

The majority of publications on software release planning focus on different kinds of mathematical models and simulations designed to create the most optimal or risk free release plans when key parameters such as resource availability, value of requirements, development effort and risk factors are known or can at least be estimated somewhat accurately (Ramesh, Cao & Baskerville 2010). The model or simulation is then used to generate one optimal release plan or a set of near-optimal release plans. Such models are called *strategic release planning models* (Ruhe & Momoh 2005). Prioritizing requirements is a central component of any release planning process. Most strategic release planning models use some variation of a well-known prioritization technique such as Analytic Hierarchy Process (AHP) (Karlsson & Ryan 1997) or Cumulative Voting (CV) (Berander & Jönsson 2006). Typically such prioritization methods are only applicable when there are relatively few requirements and the requirements can be understood based on a short description (Ngo-The & Ruhe 2008), although more complex variants claim to mitigate the scalability problem to some extent (Berander & Jönsson 2006, Karlsson, Olsson & Ryan 1997).

The validity of most of the strategic release planning models in a large-scale industrial setting is questionable. Svahnberg et al. (2010) reviewed 22 strategic release planning models. Only four of those models had been validated in large-

scale industrial use. In addition, we (and others, e.g. (Cao & Ramesh 2008)) have noticed that the model-driven approach to release planning is problematic in practice, as many software companies do not have a software development process which could be relied on to record or generate required key parameters (Ramesh, Cao & Baskerville 2010). Often requirements change so frequently that any long-term plan quickly becomes obsolete (Boehm 2000), and the aforementioned scalability problem is inherent to the strategic release planning models (or at least to the ones created so far). Agile software development methods claim to mitigate these issues by not creating detailed long-term plans and by adapting to changing customer needs and priorities when needed (Abrahamsson 2003).

Strategic release planning models have not been widely studied in an agile software development context. In 2009 we tried a plan-driven release planning approach in an agile software development context (Heikkilä et al. 2010). We found that the plan-driven approach created mathematically more optimal plans, but the differences between the optimal plans and a manual plan were quite small and the participants felt that much of the extra complexity introduced by the mathematical model was unnecessary. A release planning paper by Li et al. describes a process for applying AHP and risk analysis in Extreme Programming context (Li et al. 2006). However, the proposed process is quite generic and does not take into account the special characteristics of an XP project. Furthermore, following a highly detailed release plan is not generally considered necessary or even useful in agile software development projects; see e.g. (Boehm & Turner 2003, Highsmith 2002). Table 11.1 illustrates the conflict between the agile software development methods and strategic release planning models.

Table 11.1: The conflict between agile software development and strategic release planning

Agile software development methods	Strategic release planning models
Just-in-time elaboration of requirements	Big upfront elaboration of requirements
Informal management of dependencies	All dependencies must be made explicit
Collaboration between individuals	Mathematical optimization based on inputs
Optimizing throughput of requirements on long term	Optimizing allocation of resources on short term
Used in industry for at least ten years	Only a few models have been validated in industry

11.3 The joint release planning method

The joint release planning method is a way to plan releases in a multi-team agile software development context. A release in this context means both a test release which contains only partial functionality and a public release which con-

tains a complete set of features selected to be published for business reasons. The features included in test releases should be complete and of publishable quality. The purpose of test releases is to act as milestones for the project and to practice and test the release process to gain knowledge and insight for improvement of the process and the product. This is very similar to Microsoft's Synchronize-and-Stabilize process model first reported by Cusumano (1995).

The strategic plans for the whole development project act as input for the release planning events. The output from a release planning event acts as a starting point for an adaptive development process during the subsequent development sprints. The release plan should not be taken as an immutable plan. Figure 11.1 illustrates the relationships between the three time horizons: a development project, release projects, and iterations. A development project starts with a joint release planning event in which the first release is initially planned. The eventual first release is then followed by a second joint release planning event in which the next release project is planned. The number and length of release projects and iterations (that is, development cadence) shown in the figure is illustrative only, as the optimal number and length is context dependent. The idea of using such a multi-tier framework for managing software development was first introduced by Rautiainen et al. (Rautiainen, Lassenius & Sulonen 2002) in 2002 and similar multi-tier frameworks have since been proposed by other authors, e.g. (Leffingwell 2007, Sutherland 2005).

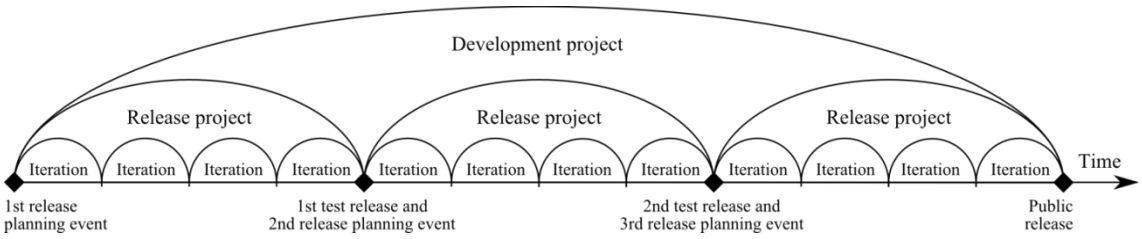


Figure 11.1: An example of development cadence

Scaling agile software development into a multi-team environment creates a need for further requirements hierarchy, as managing thousands of low-level requirements quickly becomes an unwieldy task for the product management organization (Lehtola, Kauppinen & Vähäniitty 2007). For the joint release planning method description we adopt the model proposed by Dean Leffingwell[29]. *Themes* denote strategic focus areas of a company's business. Within these, *Epics* define high-level requirements for products. Epics can be split into more detailed *Features*, which in turn can be further split into *User stories* (or simply stories). Finally, user stories are refined into development *Tasks*, which denote what needs to be done technically to implement a user story. The model your organization uses probably differs from the model proposed above. In that case try to identify the levels of your requirements hierarchy which corresponds

[29] http://scalingsoftwareagility.wordpress.com/2008/12/04/a-lean-scalable-requirements-information-model/

to those described above. For one alternative approach to requirements hierarchy and a Scrum-like process for product management, see Vlaanderen et al. (2009).

Large, complex product- and multi-team environments also create a need for additional product management hierarchy. In our joint release planning method description, we extend the roles defined in Scrum (Schwaber & Beedle 2002) with an additional *product manager* role. Scrum defines a *development team* which consists of four to seven *software developers*, a *scrum master* and a *product owner*. Software developers are responsible for creating tasks and stories, product owners assist software developers in story creation and assist product managers in feature definition, and product managers are responsible of creating and managing features and epics. Competent product owners are crucial in agile software development, as a product owner needs not only enough technical knowledge to communicate efficiently with software development, but also enough business knowledge to communicate efficiently with the business side of the organization. Unlike the other roles, product managers do not belong to any single team.

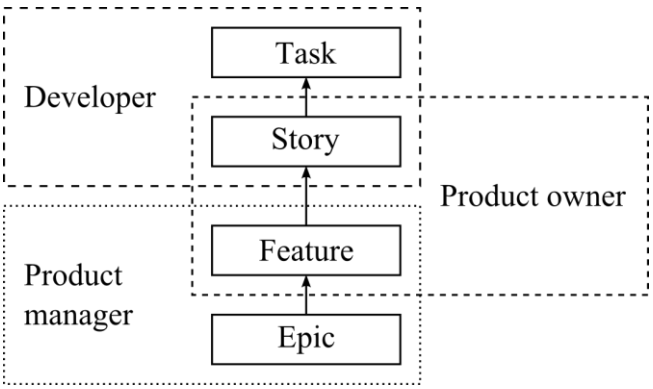


Figure 11.2: The relationships between the roles and the requirements hierarchy

Figure 11.2 illustrates the relationships between the requirements hierarchy and the roles. Depending on the organization and the type of software it develops there might be other roles such as software architect, usability expert or tester who are members of the project organization. According to agile software development principles, the team members with such skills should be embedded in the cross-functional development teams. In reality there is often a need for experts with highly specialized skills who do not belong into any single team. Reinertsen (2009) suggest several methods for utilizing such experts, but the main lesson is not to overwork the experts. You should be able to pull experts into teams to solve difficult problems on short notice, which is close to impossible if they are already working on maximum capacity.

A joint release planning event is the focal point of the method. The event should be facilitated by a person with process coaching experience, for example an experienced scrum master. During an event the whole project organization simul-

taneously plans the next release of a product. A joint release planning event consists of three segments which differ on content and purpose. The first segment contains introduction to the project and guidance for the planning, the actual planning work is done iteratively in team planning breakouts during the second segment, and the plan and risks are reviewed in the third segment. Figure 11.3 illustrates the structure of the event. Continuous improvement is an important aspect of the planning method, but the actual improvement work may be performed outside the joint release planning events.

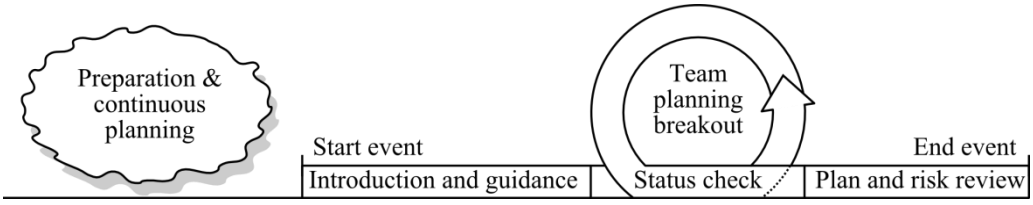


Figure 11.3: The timeline of a joint release planning event

Two other important aspects of the method are preparation for a joint release planning event and continuous planning between releases. Preparatory work is crucial for the success of the event and, considering the whole development project, the success of the product. Much of the preparatory work is related to such software product management subtopics as product roadmapping and requirements management (Weerd et al. 2006). It is important that product managers and product owners keep the backlog of epics up to date between releases. This continuous planning work is based on progress and other information from development and the market, and other information from the business organization.

The main responsibility of a software development team in a release planning event is the creation of a release project plan for their team together with the team's product owner. The team's product owner should be always available for the team to clarify requirements and accept finished stories. Scrum master is a Scrum-specific role. In addition to coaching the Scrum practices, a scrum master is also responsible for removing impediments. In a release planning event, a scrum-master is responsible for guiding teams release planning practices.

11.3.1 Preparation

The most important task for product managers and product owners in preparation for a release planning event is prioritizing and selecting the features which are included in the release planning. The list of features should be long enough to provide a good basis for planning the next release project. The features' descriptions do not need to be detailed, since product managers and product owners are present in the release planning event to provide just-in-time elaboration, and stories created based on the features will provide more concrete guidance for the developers. The list of features should be provided to the development teams well before the event to speed up the planning process during the event.

One way to discuss features in advance is using the backlog grooming practice or 5% workshop, where the development team and product owner look ahead in the product backlog to clarify and split upcoming features.

If there are any specialized roles, such as architects or user experience experts, they should also prepare any relevant materials beforehand and provide it to the development teams at least a couple of days before the planning event. The development teams should prepare by making sure that any progress information is up to date and the team's velocity for the sprints in the next release project have been calculated, taking into account any irregularities such as vacations or other planned absences of team members.

There is a practical limit of self-organization in this type of release planning, as it is not practical to have multiple teams discuss the feature assignments together. In an ideal agile development organization all teams are equally capable and assignment of features to teams is simple. However, in reality the different teams often have different areas of expertise and capabilities. Product managers together with product owners should tentatively pre-assign features to teams based on their best knowledge on the teams' expertise and capability. However, creating a bottleneck by assigning many critical features to a single team with special expertise should be avoided. Instead, other teams should be allowed to spend time to gain the special expertise and the critical features should be spread among the teams. It is important to remember that the assignments of features to teams are only tentative and the teams are free to change the assignments based on the new information that is uncovered during a release planning event.

In addition to pre-assigning the features, the features should also be tentatively prioritized before a planning event. The priority order should be per team, since the priority order should allow development teams to make scheduling trade-offs caused by inter-team dependencies. For example, a team might need to schedule a less important story earlier than a more important story if another team's very important feature has a dependency to the less important story. Again, the priorities are subject to change during the release planning according to any new information uncovered during the event.

In any non-trivial project there are more features than can be implemented in the next release project. Product management should limit the number of features they select into a release planning event to a realistic stretch goal. Including features that have no chance of being implemented in the next release project is simply a waste of effort and tracking what actually has been planned to be included in the next release project becomes more difficult. Also, lean and flow thinking suggests that focusing on maximizing capacity utilization only slows things down, like in a traffic jam on a highway during rush hour.

For the joint release planning method to work, the product management practices must support the agile development model. Traditional product management literature views development as a black box where product requirements are put in and a product matching those requirements comes out (Ebert 2009, Kahn, Castellion & Griffin 2005). In an agile software development organization, product management must be capable of adapting to changes in both market and development plans during the development of the product. The materials prepared for a release planning event by product management must take into account real development progress, which also means that development progress must be communicated to product management during release development. One example of a formalization of such a two-way communication process is the Agile Requirements Refinery described by Vlaanderen et al. in Chapter 10.

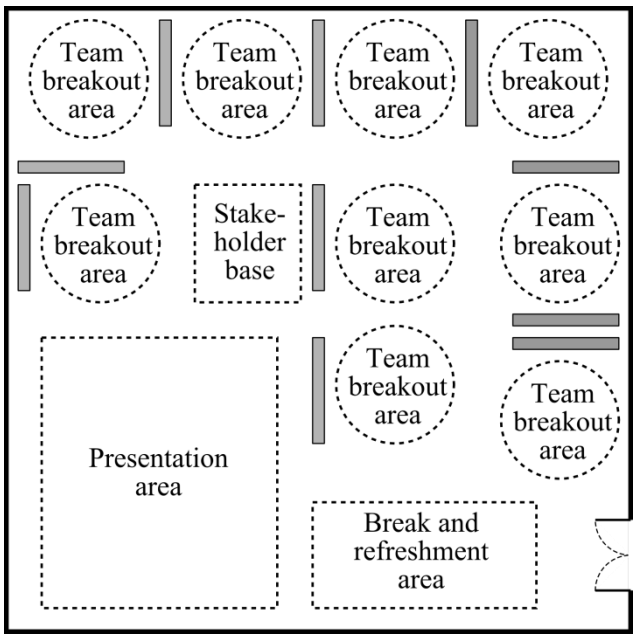


Figure 11.4: An example of joint release planning space floor plan

The large number of people creates some requirements for the space reserved for a joint release planning event. We have observed joint release planning events in several different kinds of spaces, for example in an empty exhibition hall, in a conference hall and in an empty office floor. Figure 11.4 shows an example of a joint release planning floor plan. Each team should have a team breakout area which acts as the planning area of the team. Each team breakout area should have at least one wall which can be used as a planning board for sticky notes, drawings, print-outs, etc. There should be a space to have common presentations and a space to have breaks and enjoy refreshments. Stakeholders that do not belong to a single team should be reserved a stakeholder base where they can work on their own when they are not needed by a team.

11.3.2 Introduction and guidance

The planning event starts with an introduction to the event. The introduction can contain such things as the composition of the project organization, the schedule of the event and any other information of the practical arrangements. The introduction is followed by instructions. The detail level of the instructions given to the session participants depend a lot on how familiar they are with the release planning method and other agile estimation and planning techniques. Some things that might be covered in the instructions are guidelines on how to split requirements to user stories, how to estimate user stories and how to prioritize them. In addition, the instructions should emphasize that the teams are responsible for solving dependencies and uncertainties concerning the requirements and should take an active role in engaging other stakeholders and teams for help.

Instructions are followed by vision presentations. The purpose of the vision presentations is to give the participants an overview of what the release project should accomplish. The most important vision presentation introduces the requirements for the product. A preliminary assignment of requirements to development teams should also be given. Requirements should be presented on two levels of abstraction: concepts for the whole release project on a very general level, and more precise requirements (such as features) that could be implemented in the following release project. The concepts provide a motivation for the development, while the more precise requirements guide the direction of the development. Specialized teams, such as architects or usability experts, act as internal stakeholders for the development teams. These internal stakeholders see the product from a different viewpoint and each internal stakeholder group should therefore give a vision presentation containing plans and requirements for the product from their perspective. Requirements presentations in the second and later release planning events should also contain an overview of the development project progress so far, and all vision presentations should emphasize the changes in plans since the initial release planning event.

11.3.3 Team planning breakouts

The actual planning of the release project starts after the introduction and guidance presentations. The teams break out of the general meeting area into their own planning areas, hence this part of the event is called team planning breakout. Each team should have the prioritized list of features and other requirements with the tentative team assignments. Each team starts planning from their top priority feature. With guidance from the team's product owner and, if required, a product manager or other specialist roles, the team breaks down the feature into user stories. The actual implementation scope of a feature is described as the user stories derived from it. Since features are relatively broad and abstract, the content and implementation order of the user stories is nego-

tiated based on balancing the technical constraints (such as implementation effort) of the stories and the business aspects (such as financial benefit or contract fulfillment) of the features. Any dependencies that are uncovered when the stories are created should be resolved immediately by the development teams. The stories are then scheduled into the sprints of the release project based on the estimated velocity of the development team taking into account any dependencies, critical resources and other aspects that affect the schedule.

Software developers often think in technical terms and might prefer to plan in large technical tasks instead of stories. However, stories should be written from a user's point of view, not from a technology point of view. Scrum masters and product owners should guide the developers to write stories instead of large technical tasks. Planning sprints on the task level should be done in a sprint planning event, not in a release planning event. However, sometimes planning on the task level might be necessary to properly understand and estimate a story and should be allowed in such situations.

The team planning breakout is an iterative process, since new information that affects the schedule is most likely uncovered during the breakout. In a multi-team environment there are always implementation order dependencies between the different teams. Whenever there is a scheduling conflict between two teams caused by a dependency, the development teams and possibly product managers should resolve the conflict by identifying the most beneficial schedule considering the whole release project. Identified risks and impediments should be, if possible, handled immediately by the development teams. Failing that, the risks and impediments should be recorded and dealt with in the next status check. The later in planning a dependency is resolved, the greater the possible effect may be to the team's plans.

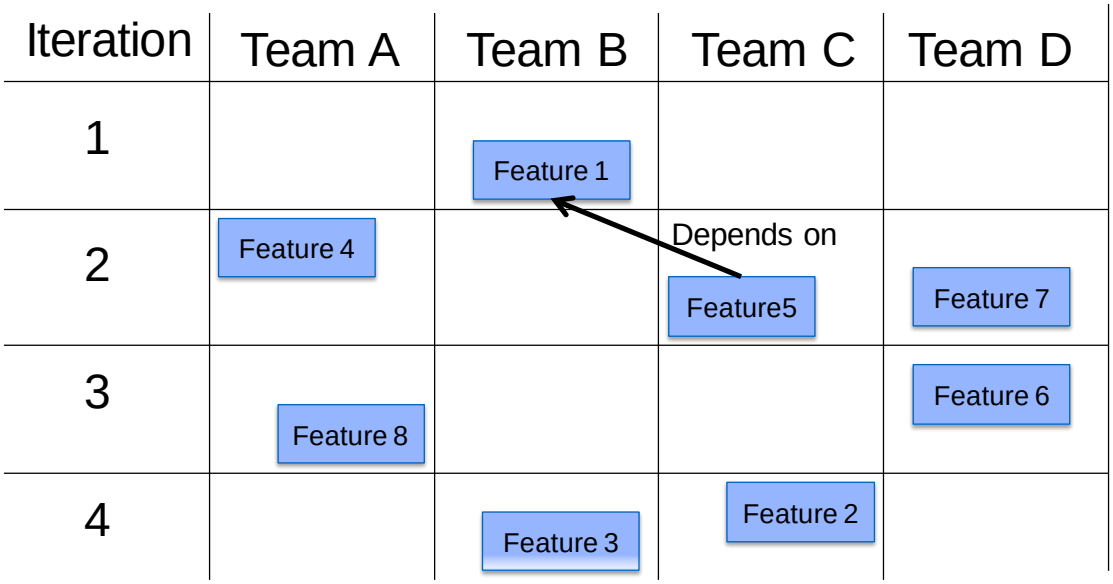
Development teams invest lots of effort into creating their release plans during the release planning event. However, the teams should understand that planning is an iterative process and plans may need to be changed based on new information that is unearthed during the event. For example, a previously undetected dependency might cause re-scheduling of user stories or the priorities of features might change based on new information revealed in the event.

Internal stakeholders should be available at the planning event and prepare materials in advance. In case there are specialized internal stakeholder groups there should be a representative from each group in the planning event. The representatives should be available to clarify issues related to their specialty areas. Any materials related to the specialty areas should be available before and during the release planning event.

It is crucial for the success of the method that the developers understand they are responsible for communicating with the members of other development

teams instead of asking indirectly from the team's product owner. Direct communication is more efficient and accurate. However, communication practices should still be agreed so that the teams' planning is not interrupted constantly by very active communication. For example, teams can agree on spending some time planning together for a requirement that has a lot of dependencies between two or more teams.

The overall progress of the planning should be tracked and communicated in a way that allows all participants to see the progress. One way to accomplish this is a visual traceability matrix where each team marks which features they are going to implement and when. The visual planning board shows the estimated start and end points of working on a feature. In addition, each story and feature should have a unique identifier which helps tracing dependencies between stories and links stories to the related feature. The figure below shows an example of a traceability matrix.



Planning status should be checked frequently and in a lightweight way. Release planning during the team breakout is intensive work for the development teams. The teams should not be interrupted frequently by status checks. On the other hand, too infrequent status checks may reveal problems too late. Therefore status checks should be lightweight and frequent. Instead of all developers of all teams, only a representative of each team should participate while the rest of the team keeps on planning. In the status checks, a representative of each team shortly reports how their planning has progressed and how much time is still needed, what their impediments are, and what dependencies, if any, they have discovered to other teams that need to be resolved. Representatives of the other teams might notice undiscovered dependencies which should be brought up and then resolved during the following team breakout. The impediments should be tackled and resolved by a suitable set of stakeholders during the following team breakout or as soon as possible. The facilitator or coordinator of the event can

use the status checks to judge how much time is still needed for the planning and use this information to flexibly plan how the rest of the planning event should proceed.

11.3.4 Plan and risk review

The final plan review starts with a review of the plan of each development team. A team member presents the team's plan for the next release project on a general level and risks associated with the team's work. Other participants are free to ask questions and comment the plan. Any previously unidentified risks are also recorded. The next step is the processing of the risks that have surfaced during the team planning breakout or the final plan review. The facilitator presents the risks one at a time and each risk is then either accepted or somehow mitigated.

11.3.5 Monitoring and steering a release projects

At some point of the release project, typically sooner than later, the progress of the release project starts to deviate from the release plan. The development teams are responsible for keeping their progress information up to date. Typically updating the status information of the team's stories at the end of each sprint is enough. The most important consumer of the teams' status information is the product management organization. If development is lagging behind schedule, product management is responsible for making scope or schedule changes to get the project back on track. Product management should also start preparing for the next release planning event right after the previous release planning event has concluded, and the plans need to take into account the current progress of the release project. Delays in one development team may also affect the schedule of the other teams via dependencies between stories. There should be one easily accessible location that contains up-to-date story development progress and dependency information. Depending on the organization, this location might be a physical storyboard located in a central place, a spreadsheet file in a shared folder, or a more sophisticated electronic tool.

11.3.6 Continuous improvement

Retrospectives (or reflection) are an important part of agile software development where the ideal is continuous improvement. Each organization has a unique context and the joint release planning method should be continuously improved to better fit into that context. While there are many ways to implement continuous improvement, the most important aspect is the feedback from the participants of the release planning event. A survey can be conducted after an event or some participants can be interviewed to gather feedback. One option that combines feedback and improvement suggestions is a release planning retrospective workshop, where representatives from all teams, from product management and possibly other stakeholders should participate. First, issues regarding the release planning method are gathered. Second, the participants

break out into teams that each have one or a few issues to solve. Third, each team presents its results to the other participants. As in any process improvement effort, it is also important to follow up on how the proposed improvements work in practice. For another approach to release retrospectives see (Maham 2008).

11.4 Motivation

In this section we present several reasons why you should try the joint release planning method. First, we claim that joint release planning is a cost-effective way to perform release planning in an agile organization. Second, we claim that joint release planning helps to uncover problems that otherwise might have been found too late and also helps to solve those problems. Third, we claim that joint release planning mitigates the problem created by component teams, if you for some reason still have to use such teams.

11.4.1 Joint release planning is cost effective.

The immediate cost of a joint release planning event might seem prohibitive. For example, a two-day event with 50 software developers equals 800 man-hours of software development effort. However, we claim that the knowledge created and shared by the interactions during a joint release planning event reduces wasted effort during the release project more than enough to compensate the effort spent in the release planning event. First, the teams would have needed to create some kind of release project plan nevertheless or risk working on less important or even wrong features. If each team creates a release plan in isolation there is a high risk that problems caused by unaccounted for dependencies surface during the release project. Second, the simultaneous planning actually decreases communication overhead, since most of the project organization is easily accessible during the planning event. Especially product managers might be hard to reach by the software developers, since product managers spend much of their time with external (to the project) stakeholders such as customers, the sales organization and upper management (Gorchels 2003).

11.4.2 The release planning method helps uncover and solve potential problems.

During a release planning event any participant can raise an issue regarding the development project, regardless if it is related to a team, a requirement, the release project or the whole development project. Since the participants represent both business and development, they have sufficient knowledge and authority to also solve the problems immediately. For example, developers might question a technology-related presumption that product management has made, or product managers might question whether some functionality needs to be implemented in-house or whether it could be acquired. This can be seen as a strength

of the joint release planning method compared to more formal and hierarchical methods in which the release plan is created by a project manager and development tasks are simply fed to the developers, e.g. traditional waterfall or RASORP (Ngo-The & Ruhe 2009).

11.4.3 Release planning mitigates problems resulting from component teams.

One of the core principles of Scrum is the use of cross-functional feature teams for software development. With feature teams, dependencies between development teams are on feature boundaries. Sometimes, for example for historical reasons, component teams are used even when the project organization can be otherwise characterized as agile. With component teams the dependencies between development teams are on component boundaries. Typically there are a much greater number of dependencies between components than between features. Component teams also cause handovers which in turn increase the need for communication between teams. The joint release planning method mitigates the problems resulting from component teams by enabling direct and effortless communication between all development teams during a joint release planning event. However, component teams still increase the importance of dependency management and the effort spent on dependency resolving.

Chapter 12: Kanban for Software Development

Kristian Rautiainen

Kanban for software development has been slowly gaining popularity during the last part of the first decennium of this millennium. It could be used, e.g., for managing the portfolio of work of a project with one or several teams participating. In this chapter we shortly present the basics of Kanban and Kanban for software development and revisit the example in Section 9.2 to show how we could manage multitasking using a Kanban board.

12.1 Definition of Kanban

Kanban is the Japanese word for "signboard" or "billboard"[30]. The term has been used in JIT (Just-In-Time) manufacturing, most famously in the Toyota Production System (TPS), denoting a signal card that triggers the production or moving of parts in a pull production system (actually the whole system is often called Kanban). Kenji Hiranabe summarizes the properties of the original TPS Kanban concept in his InfoQ article[31]:

Table 12.1: The properties of the original Kanban concept in TPS

Physical	It is a physical card that can be held, moved, and put into or onto things.
Limits WIP	It limits WIP (Work-In-Progress), i.e. prevents overproduction.
Continuous flow	It notifies needs of production or parts before they run out of stock.
Pull	The downstream process pulls items from the upstream process.
Self-directing	It has all information on what to do and makes production autonomous in a non-centralized manner and without micro-management.
Visual	It is stacked or posted to show the current status and progress, visually.
Signal	Its visual status signals the next withdrawal (of parts from storage) or production actions.
Kaizen	Visual process flow informs and stimulates Kaizen, continuous improvement.
Attached	It is attached to and moves with the supplied physical parts.

[30] http://en.wikipedia.org/wiki/Kanban
[31] http://www.infoq.com/articles/hiranabe-lean-agile-kanban

12.2 Kanban for software development

Kniberg and Skarin (2010) characterize the minimum for what can be called Kanban for software development:

- Visualize the workflow (write each item on a separate card and put it on the wall, use named columns to illustrate the position of each item in the workflow)
- Limit WIP (write explicit limits on how many items at a time can be in a certain workflow position)
- Measure the lead time (the average time to complete one item) and optimize the process to make the lead time as small and predictable as possible

Co-located Scrum teams who use a so-called Scrum wall for sprint task visualization usually do the first of the above list, but not necessarily the others. Sometimes that is erroneously called doing Kanban. Kniberg and Skarin (2010) summarize a comparison of Scrum and Kanban (see Table 12.2). But remember that while a comparison like this can be done, we are still talking about two *tools* for managing work. If you know how to use the tool for managing your work, you are probably successful. Neither is better or worse than the other, except maybe for some particular contexts, but we will not go into that discussion here.

Anderson (2010) extends Kniberg's characterization of Kanban to 5 properties:

1. Visualize workflow
2. Limit work-in-progress
3. Measure and manage flow
4. Make process policies explicit
5. Use models to recognize improvement opportunities

For visualizing the workflow Anderson suggests mapping the value stream, especially showing the interactions, handoffs, queues, buffers, waiting, and delays involved. Measuring and managing flow focuses on keeping the work moving at a steady flow through the value stream and concentrating effort on improving. Concentrating on flow instead of waste removal may be the key to avoid some *Lean and mean* anti-patterns and dysfunctions according to Anderson. Making process policies explicit encourages looking at the work process as a set of policies rather than just a workflow. Using models to recognize improvement opportunities means taking a systematic and scientific approach to improvement.

Table 12.2: Similarities and differences of Scrum and Kanban

Similarities	Differences	
	Scrum	**Kanban**
Both are Lean and Agile	Timeboxed iterations prescribed	Timeboxed iterations optional
Both use pull scheduling	Team commits to a specific amount of work per iteration	Commitment optional (service-level agreements often used)
Both limit WIP	Uses velocity as default metric for planning and process improvement	Uses lead time as default metric for planning and process improvement
Both use transparency to drive process improvement	Cross-functional teams prescribed	Cross-functional teams optional, specialist teams allowed
Both focus on delivering releasable software early and often	Items must be broken down so they can be completed within one iteration	No particular item size is prescribed
Both are based on self-organizing teams	Burndown chart is prescribed for progress monitoring within an iteration	No particular type of diagram is prescribed
Both require breaking the work into pieces	WIP limited indirectly per sprint	WIP limited directly per workflow state
In both, the release plan is continuously optimized based on empirical data	Estimation prescribed	Estimation optional
	Cannot add items to ongoing iteration	Can add new items whenever capacity is available
	A sprint backlog is owned by one specific team	A Kanban board may be shared by multiple teams or individuals
	Prescribes 3 roles, product owner, scrum master, and development team	Does not prescribe any roles
	A Scrum board is reset between iterations	A Kanban board is persistent
	Prescribes a prioritized product backlog	Prioritization is optional

Hiranabe shows an example[32] of using a Kanban board in a waterfall development model, but with a flow. While the project he describes has different sequential process areas, such as design, development, validation, etc., the Kanban cards move between the processes, not as a big group (large batch size), but one at a time like in the one-piece-flow of manufacturing (small batch size). He describes the whole system as *a stable, sustaining phase in a product's lifecycle, managed in a waterfall state transition model with a flow*. This probably means that the particular product under development is in a mature state of its lifecycle and less innovative and more predictable work is performed. But it still might mean that integration is done often, putting pressure on the technical development practices and skills, as well as the infrastructure (build servers, version control system, etc.). These have always played an important role for any organization in the pursuit of being agile or flexible, as embodied by the practices of e.g. eXtreme Programming. For this the Kanban board can prove to be a very helpful driver for improvement. Think, for example, that integration has a problem and work starts to pile up in the workflow state before it. The Kanban board will make it painfully visible and the WIP limits will stop work sooner rather than later, so that everybody's attention is at solving the problem in integration (or rather the root cause(s) that lead to the problem). This may or may not speed up problem solving, but it most certainly will prevent a massive queue of code piling up, waiting to be integrated, which, if left to be piled up, might cause new problems which would be harder to solve causing a negative spiral. And the solution to the problem should lead to process improvement, which could speed things up in the future. In this way Kaizen is subtly performed.

As Kanban can be deployed to the current process of an organization by mapping the workflow or value stream, it provides for an evolutionary improvement path towards a leaner organization. In contrast, agile transition often involves a revolution in the ways of working, easily causing more resistance to change. A case example from (Kniberg & Skarin 2010) shows that the evolution can be quite fast. In only three months a system administration team turned from a bottleneck team that everyone was complaining about to the top-three list of positive experiences as voted in the company retrospective.

Most of the examples (in blog posts, slide sets, or other sources) on Kanban software development seem to be of software maintenance, or the maintenance team. One simple explanation for this is that it would be quite challenging for a maintenance team or a development team who is also responsible for maintenance of a product to plan and commit to an iteration, because of the nature of maintenance work. While you could reserve 40% of your team capacity (in the case of a development team also responsible for maintenance) in an iteration for

[32] http://www.infoq.com/articles/hiranabe-lean-agile-kanban

maintenance tasks, the exact portion of capacity will vary, meaning that the capacity for doing development work will vary. By removing the iteration timebox boundaries you can more flexibly take on work and the Kanban board will help you do it in a systematic and responsible way, paying attention to flow, instead of just thrashing between tasks. The visualization of the process also helps when discussing urgent requests and the trade-offs involved; if work on the urgent request is started immediately, what should be aborted instead?

12.3 Revisiting controlled multitasking with Kanban board

The example in Section 9.2 showed how floating backlogs helped Teams A and B manage complex multitasking. Here we revisit the same example and show what it could look like if the teams were using a Kanban board instead. Figure 12.1 shows how the Kanban board looked at the time just before the example starts. In this example work is already underway on some work items. Otherwise the background story is the same as in Section 9.2 and we use the same points in time from that example (the iteration planning meeting times) to show snapshots of how the Kanban board looks in this example.

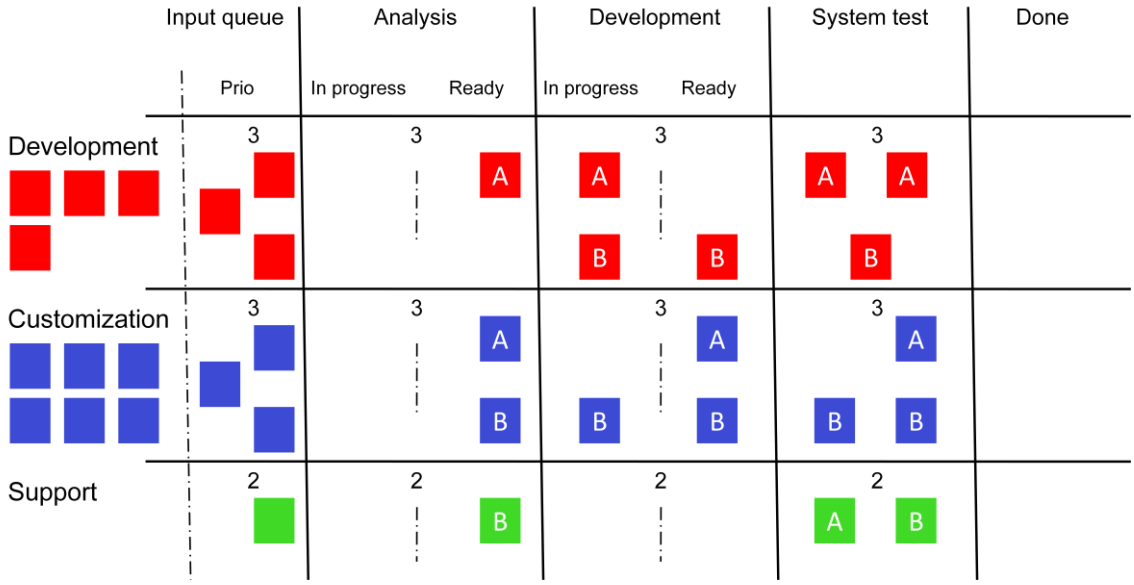


Figure 12.1: The Kanban board of Teams A and B before the beginning of the example

The Kanban board in Figure 12.1 should be interpreted in the following way. The lanes represent the three activities (thus multitasking) Teams A and B are working on; development, customization, and support. The squares represent work items (stories or features) for the activities (red for development, blue for customization, and green for support). A work item marked with the letter 'A' means that Team A is working on it and 'B' means that Team B is working on the work item. The columns model the work process of the teams. Each column has a Work-In-Progress (WIP) limit per lane denoted by the numbers in the

board. There cannot be more work items in each area of the board than the WIP limit number.

The input queue contains all the work items that are currently planned, e.g., for the next release of a product, or in the case of support work items all existing requests. From those, only 3 work items each for development and customization, and 2 work items for support can be chosen as *top priority* to be pulled for analysis when the WIP limits permit it and the teams have time for those work items.

Both the Analysis and Development column has been split into two parts: *in progress* denotes work underway and *ready* denotes work queuing for the next *phase* of the work process. These both parts together cannot exceed the WIP limit of the whole area. As we can see in Figure 12.1 system testing is progressing at maximum WIP limit and there are 3 work items queuing for system testing. This could mean that system testing capacity should somehow be improved as the next improvement effort, especially since the ripple effects can be seen in the analysis column as 4 queuing work items. This issue would probably be the topic of the daily standup meeting at the Kanban board.

Now, let's jump forward in time to the first step of the example in Section 9.2. Instead of pulling a lot of work items into their backlogs, Teams A and B now check the situation on the Kanban board and act accordingly. Figure 12.2 shows the snapshot of the Kanban board at this point in time.

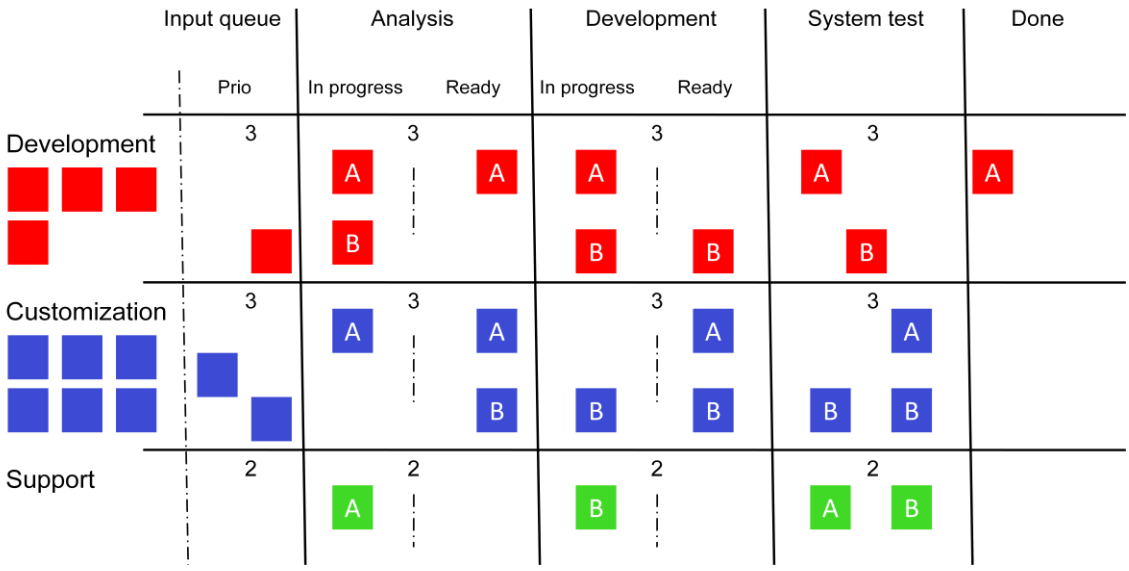


Figure 12.2: Snapshot of the Kanban board at the time of step 1

As we can see in Figure 12.2, Team A has managed to finish system testing on 1 work item, which is now done. As Team A feels it has capacity to start new work and the WIP limit for analysis allows it, Team A pulls 1 development work item and 1 customization work item for analysis, which means that the team (or some team members) analyzes what needs to be done technically to realize the work

item as working software (similar or equal to splitting a story into tasks). At this time the team may need additional information from the person(s) who own(s) the work items in question. Team B has started development work on 1 support work item and is struggling with system testing. Therefore Team B only feels comfortable in pulling 1 development work item for analysis. Team B could have chosen the only support work item, but they felt more familiar and comfortable with the development work item. Team A feels it can still manage to pull the only support work item for analysis (Team A has only 1 work item in progress in development + system testing work), which completes the description of what can be seen in Figure 12.2 compared to the situation in Figure 12.1. Since Team A is doing well and Team B is struggling with system testing, both teams agree that a senior system testing specialist from Team A will help Team B with their system testing for the next couple of days. While this will slow down the work of Team A, it should improve the overall flow of work. At least both teams and the managers think it is worth trying.

Step 2 of the example in Section 9.2 follows fairly close in time to step 1, but a lot has still happened on the Kanban board in our example (Figure 12.3).

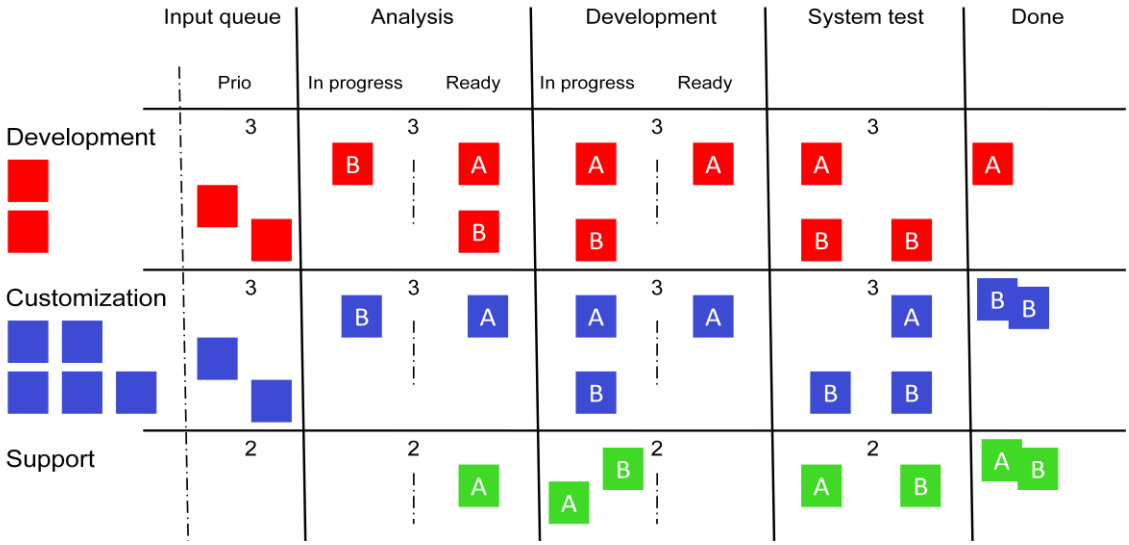


Figure 12.3: Snapshot of the Kanban board at the time of step 2

The help of Team A's senior system testing expert made it possible for Team B to finish testing 2 customization work items and 1 support item. Team B also eagerly pulled more work items to system testing, and as we can see in Figure 12.3 Team B still has most of the system test work items, but now they are better equipped to deal with them. Team A's testing has stalled a bit due to decreased testing expertise and capacity, but some of the other team members are learning new skills so that they can help with system testing, while some work items are queuing. Three new support work items had appeared in the input queue since step 1, and as we can see two of them are already in progress in development and the remaining one is ready in analysis. One new work item from both development and customization has been pulled into analysis by Team B.

Moving on to the time of step 3 of the example in Section 9.2 a lot has happened on our Kanban board (Figure 12.4).

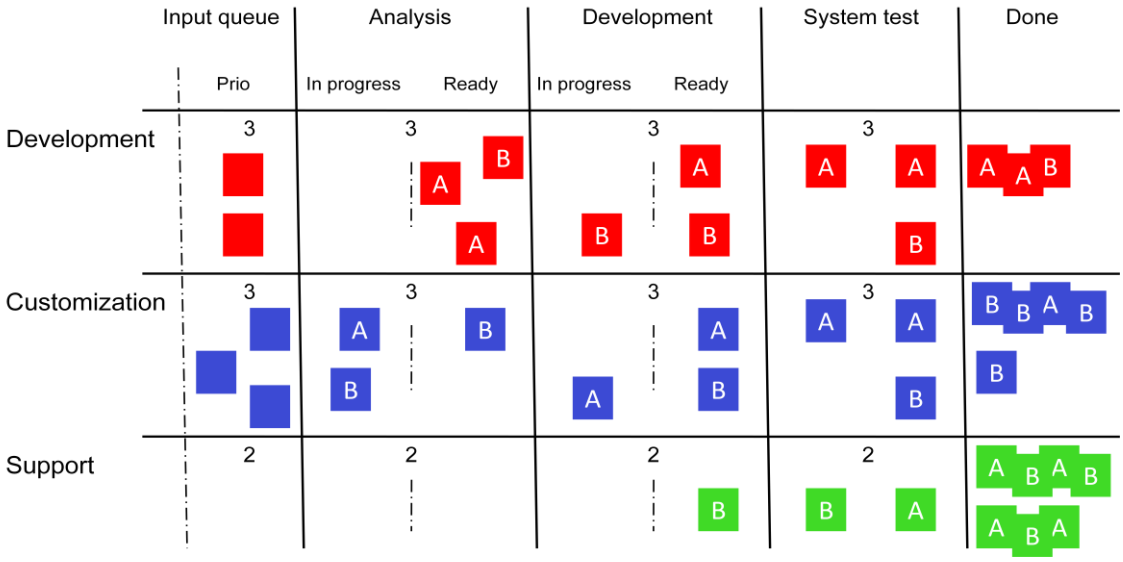


Figure 12.4: Snapshot of the Kanban board at the time of step 3

One clear observation from Figure 12.4 is that Teams A and B are very responsive to support requests. In fact, customer satisfaction of the organization has increased due to this responsiveness. Three new support work items had appeared since step 2 and they are almost done, along with all the other support work items. System testing is still hard work causing queues both in the development column and indirectly in the analysis column. As senior management has now seen the evidence visually from the Kanban board, they have reacted to increasing system testing capacity by initiating a system testing training program and hiring one new senior system test expert for the teams to share as seen fit.

Figure 12.5 shows a snapshot of the Kanban board at the time of step 4 of the example in Section 9.2. Most work items have been done, among them all the support work items, even the 2 new ones that appeared since step 3. As you may wonder why there are no new work items in the input queue to pull from, this is because we have chosen to leave them out in this example for the sake of clarity. In reality there would naturally be a long list of potential work items to prioritize and pull from. But without a proper process, such as the requirements refinery in Chapter 10, it might in fact be difficult for product management to keep up with the development teams, once these get their development processes in shape. However, in this example we have left the complexities of architectural choices and requirements refining behind the veils of input queue and analysis (much like Scrum leaves them behind the roles of product owner and empowered, self-organized development team). In reality, these remain very hard to do successfully, no matter what the process is.

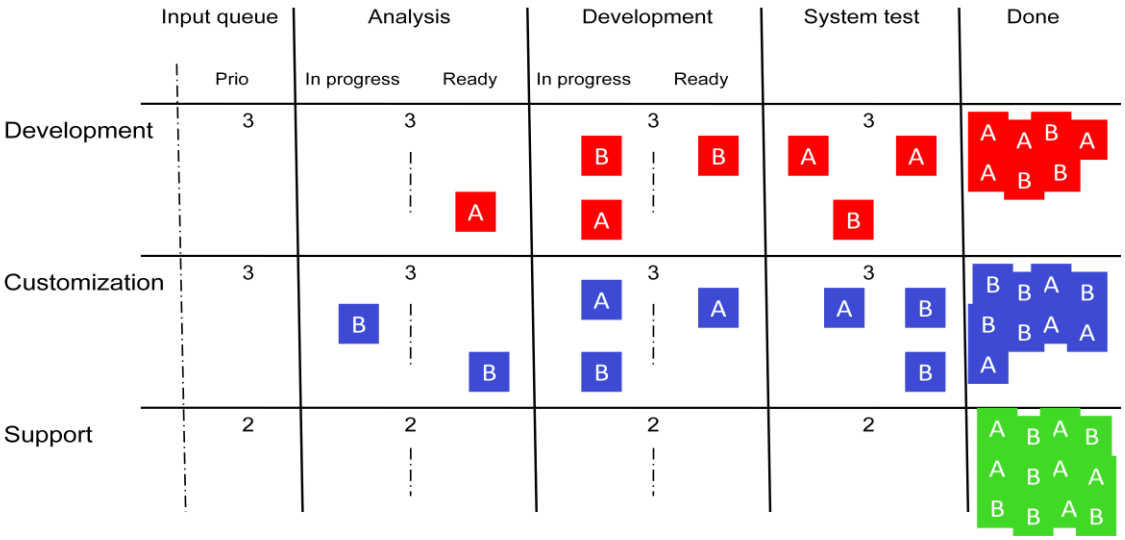


Figure 12.5: Snapshot of the Kanban board at the time of step 4

By the time of step 5 of the example in Section 9.2 all work for the development and customization releases has been done. We have now shown two ways to tackle the complex problem of multitasking. In both ways of working, continuous planning is needed to get the right work items into the prioritized input queue. When using timeboxing, as in the earlier example, the first thing you should consider is synchronizing the cadence of different activities to prevent the situation from escalating to an even more complex one. In Kanban software development strict timeboxing is not necessarily used. However, some kind of cadence, at least for (re-)planning is recommended.

If you think that Kanban for software development might be something you want to try, you should check out the following books for more information; (Kniberg & Skarin 2010, Anderson 2010, Ladas 2008).

Chapter 13: Requirements for a Backlog Management Support Tool for Agile Product and Portfolio Management

Jarno Vähäniitty & Ville Heikkilä

The earlier chapters in this part have described product and portfolio management as they should be understood in agile software development. This chapter summarizes the key requirements for a backlog management support tool as posed by the presented frameworks. The discussion has been structured according to the Cycles of Control framework as *Product management* (Section 13.2; incl. *Roadmapping* and *Release planning*), *Development portfolio management* (Section 13.3) and *Daily work* (Section 13.4). Before going into the requirements themselves, we first discuss the scope of the discussion in Section 13.1.

13.1 Scope of the discussion

The intent here is to focus on the essentials and present a blueprint for backlog management tool support in the context of agile product and portfolio management as guided by the understanding collected in this book regarding these areas, and their interplay with daily work. Requirements for supporting product portfolio management have been left out of the scope of this book. For the rest of the cycles, we focus on only those requirements that are crucial for supporting the frameworks described in Part III of this book. This means that requirements that may be important for backlog management, such as whether the tool should be physical (that is, whiteboards, post-its, etc.) or electronic, and whether an electronic system should integrate with version control and issue tracking, user rights management and so on are simply omitted. Also, details concerning iteration management are skipped as well, since most of existing tool support – whether electronic or physical – can handle single iterations quite well, provided of course that product and release levels are in order.

In this chapter, requirements are expressed in the so-called canonical user story format; each user story is followed by a short version of its name in parenthesis.

Those user stories that are to a reasonable degree supported by the latest version of Agilefant available at the time this book is published are denoted with a small Agilefant logo ( ) next to the user story name. For those user stories that are only partially supported, footnotes are used to denote the shortcomings of the current (version 2.0.4) implementation. Where appropriate, the current implementation in Agilefant is illustrated by a screenshot.

Also, note that the user stories in the following sections are by no means an exhaustive list! Rather, they are meant as a minimal checklist to keep in mind when looking for or designing a backlog management tool suitable for supporting long-term product and portfolio management as represented by the frameworks presented in Part III of this book.

We have since 2008 been collecting a list of backlog and project management tools to

www.tinyurl.com/biglistoftools

The list is freely editable, so feel free to add the tool you are using! Also, we welcome you to do a self-evaluation of your tool against the requirements set forth in this chapter.

13.2 Product management

From the perspective of backlog management, product management consists of Roadmapping and Release planning and monitoring. These are addressed in Sections 13.2.2 and 13.2.3 below. But before delving into specific requirements concerning them, we discuss the implications of linking product management and agile software development from the perspective of backlog management tool support in Section 13.2.1 below.

13.2.1 Work item hierarchy and the backlog

As explained in Section 7.3.2 (*Splitting work items and traceability*) and Section 7.3.3 (*From strategy to action and back again*), we claim that a prerequisite for long-term planning with backlogs is the ability to create and maintain structures of hierarchical work items.

This yields the following requirement:

> **As the product owner, I want to be able to create and manipulate work item structures with unlimited hierarchy in order to enable just-in-time backlog elaboration, estimation and prioritization (Story tree)**

Agilefant's implementation of *Story tree* is illustrated in Figure 13.1 below:

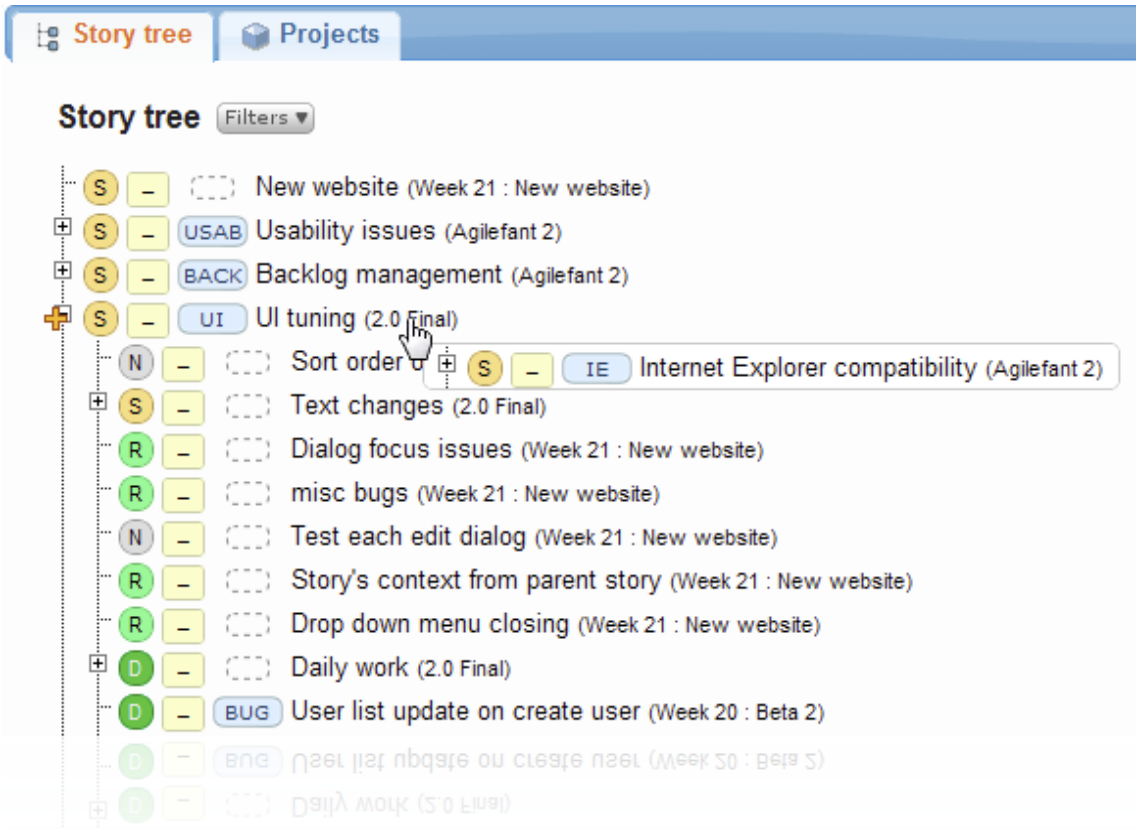


Figure 13.1: Story tree for creating and manipulating work item hierarchies in Agilefant 2.0.4

However, viewing the backlog as a prioritized list is still needed for prioritization:

> **As the product owner, I want to view the backlog as a rank-ordered flat list in order to accurately prioritize work items in preparation for release and iteration planning (Flat list view)**

Agilefant's implementation of *Flat list* is illustrated in Figure 13.2 below:

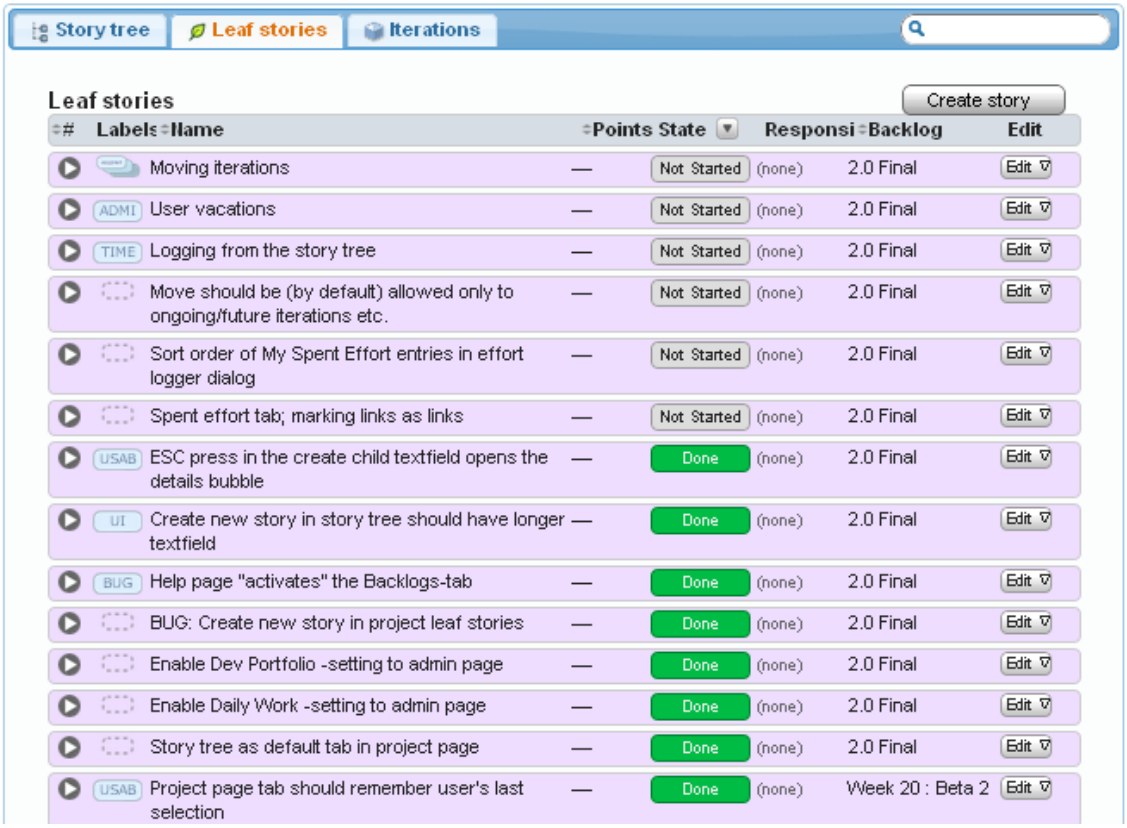


Figure 13.2: Flat list for work item prioritization in Agilefant 2.0.4

As the team mostly works with flat lists, they still need to be aware of the work items' context. Thus, we end up with the following user story:

> **As a developer, I want to be able to see the high-level context of work items to help me ask the right questions and make the right design decisions (Context)**

The implementation of *Context* in Agilefant 2.0.4 is illustrated in Figure 13.3 below.

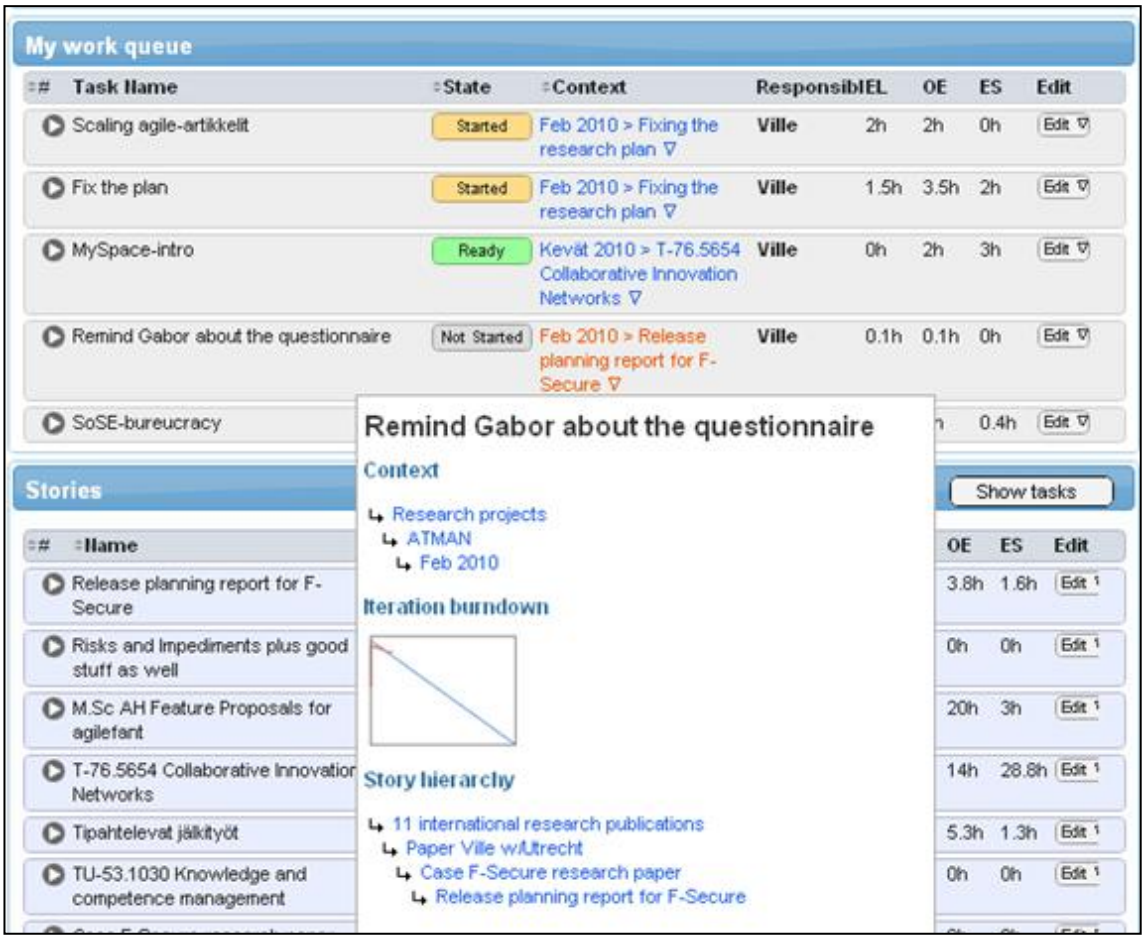


Figure 13.3: Context pop-up from a flat list in Agilefant 2.0.4

13.2.2 Roadmapping

Based on the definitions of *roadmap* and *roadmapping* as described in Section 7.3 (*Linking agile with long-term product and release planning*), we have derived the following user stories:

> **As the product owner, I want to define one or more releases of the product, complete with release dates in order to have containers into which I can schedule work items (Releases)**

And in more detail:

> **As the product owner, I want to schedule work items into releases in order to express the long-term plans of where the product is going on a more tangible level than the product vision (Scheduling [33])**

In Agilefant 2.0.4, the view that in principle would be best suited for *Roadmapping* would be the Leaf stories tab on the Product level (Figure 13.4):

[33] This can to some degree be achieved in Agilefant 2.0.4; see restrictions on the following page.

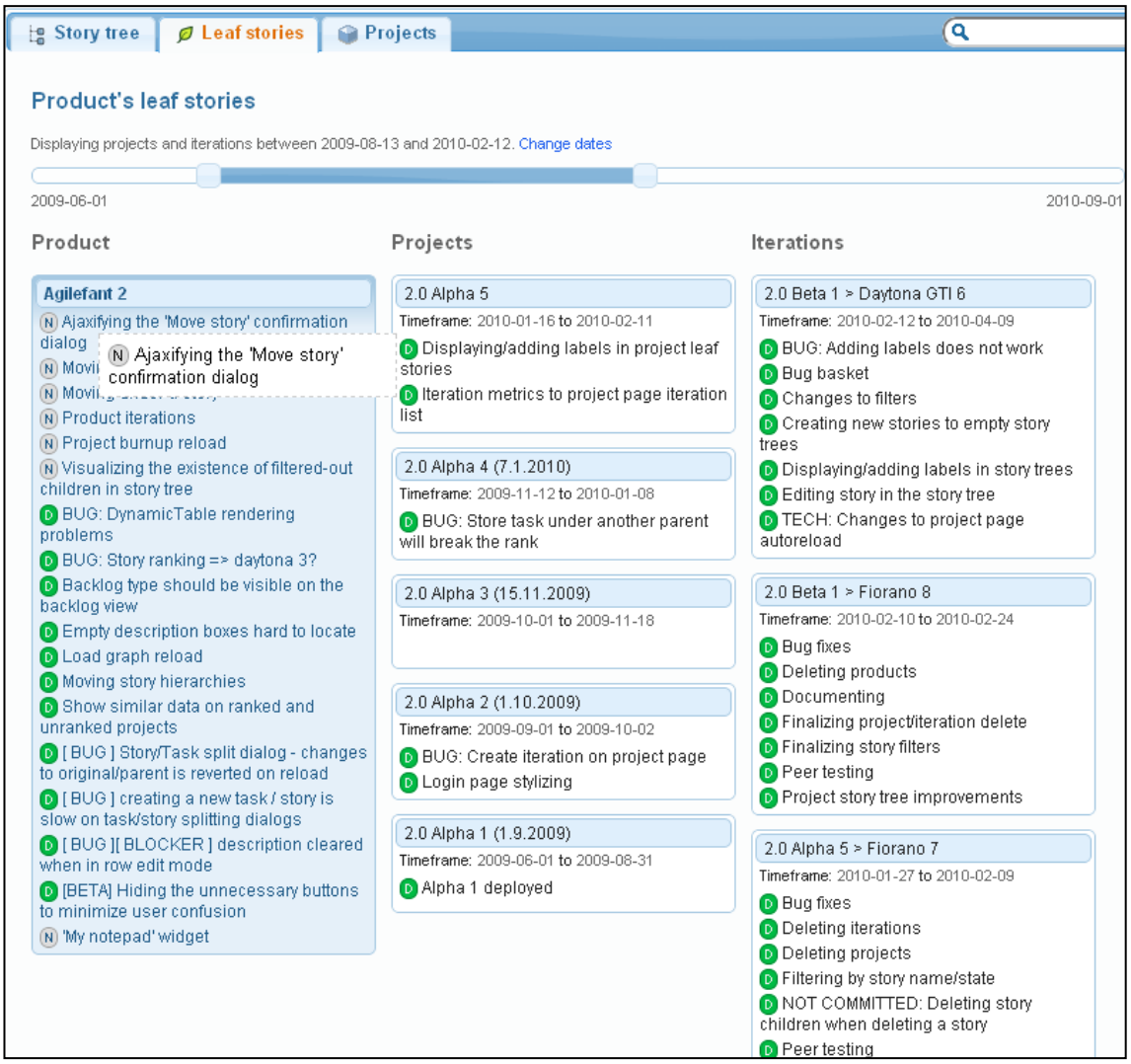


Figure 13.4: Agilefant 2.0.4 allows for Roadmapping with leaf stories

However, the current implementation only displays leaf stories (that is, those work items that have no children), and thus does not fully support work item scheduling as we currently understand that it should be done[34].

The restriction of forcing to plan future releases and iterations in terms of leaf work items seems to apply to most of the electronic backlog management tools currently available. If you are aware of a tool that does a good job with respect to this requirement, let us know.

The final major requirement for backlog management tool support regarding roadmapping as we understand it is the capability to display the resulting roadmap (or parts of it) visually and with a degree of detail that suits the audience in question. We formulate this as follows:

> **As the product owner, I want to be able to communicate the long-term plan of the product visually and on a suitable lev-**

[34] This can to some degree be worked around by using the Story tree view on the Product level to schedule work items with children into upcoming releases.

el of granularity for different stakeholders; developers, business people and customers alike (Roadmap visualization)

13.2.3 Release planning and monitoring

Based on the discussion of release planning in Section 7.1 (*What is release planning?)* and Chapter 11, we have defined the following user stories for release planning and monitoring tool support:

As the product owner, I want to express a preliminary release plan by allocating features and stories into the upcoming iterations (Iteration level scheduling [35])

To support the release planning method described in Chapter 11, the following user story is required:

As the release planning event facilitator, I want to print out a subset of features and related user stories (Backlog-to-paper)

As for monitoring the progress of a release, the basic requirement can be posed as follows (refer to Figure 7.4 *The ATMAN framework for linking daily work with product and business goals* and Figure 8.3 *From investment levels to product/business area vision, goals, actions – and back again*):

As the product owner / product manager / business owner, I want to monitor the progress of a release in terms of goals of the level I can understand (Release monitoring)

A basic metric that can be used to monitor the progress of a release is comparing what has already been done to the current total scope of a release:

As someone responsible for the release, I want to see a burn-up graph that compares the total amount of done story points to the current scope of the release (Release burn-up [36])

The release burn-up (called 'project burnup') in Agilefant 2.0.4 is described in the paragraph below.

[35] In 2.0.4, only leaf stories (that is, work items that do not have children) can reside in an iteration. Thus, if a feature has already been (at least partially) split into stories, it cannot as such be scheduled into an upcoming iteration.

[36] Agilefant 2.0.4's burn-up is based on comparing the amount of done story points to total scope calculated from the leaf stories only; thus, it does not account for only partially split work items. A better burn-up would display two scope graphs, or, use only that number which is the greater as the total scope.

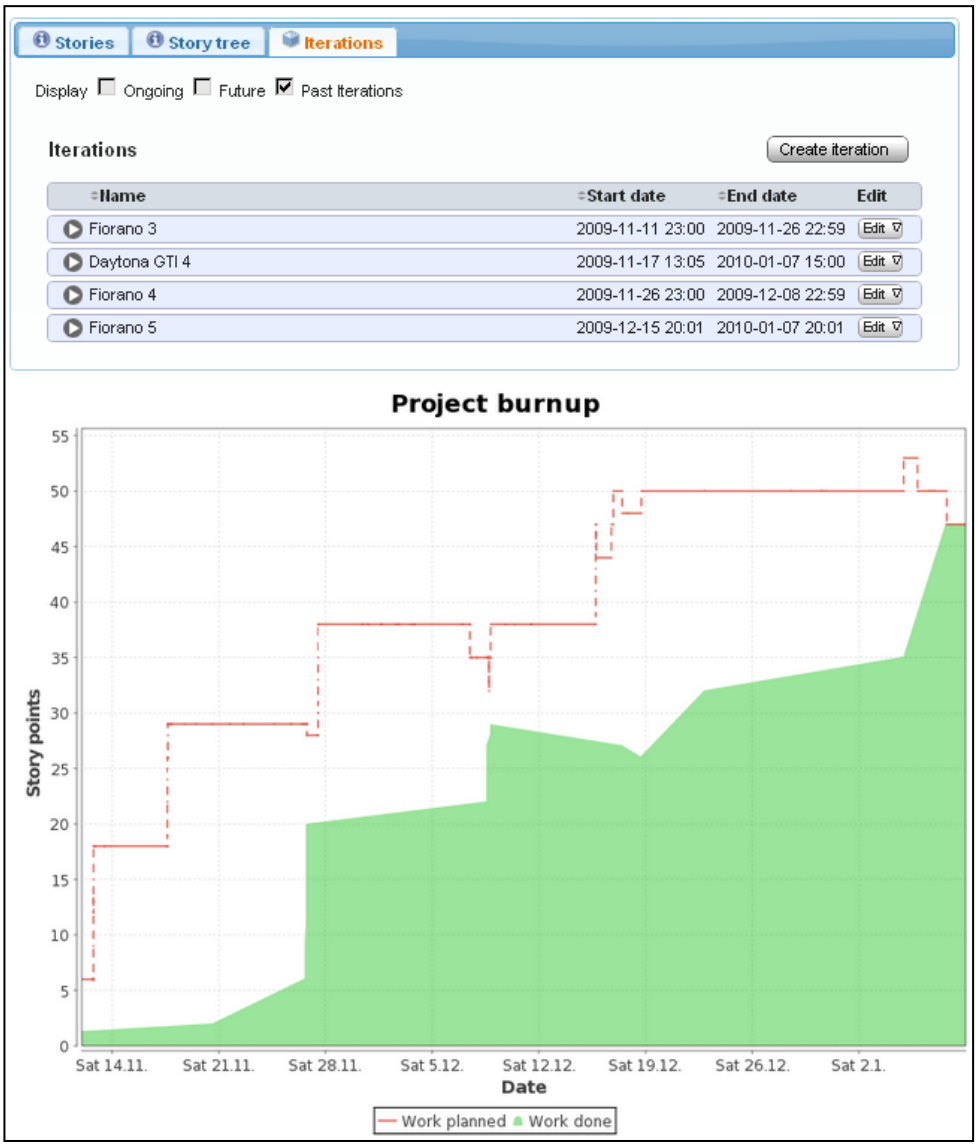


Figure 13.5: The project burn-up and the iterations of a completed release in Agilefant 2.0.4

The burn-up in Figure 13.5 contains two graphs to show the planned scope and progress of the release as the function of time. The red line denotes the scope of the release as the initial story point sum from those work items that are planned to be completed in the release (reside in the backlog of the release, or in the backlog of any of its iterations). 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 or are moved out of the scope of the release (for example, it is realized that an item is no longer needed or cannot be completed in time), the two graphs get closer, and eventually meet.

13.3 Development portfolio management

The requirements for supporting development portfolio management have been divided into two categories: *Portfolio overview* and *Load management*. These are described in Sections 13.3.1 and 13.3.2 below.

13.3.1 Portfolio overview

The main thing in development portfolio management is to get an overview of what is going on, who is doing what, and see (even on a coarse level) what the status of the ongoing activities is. This yields the following user stories:

> **As a member of the portfolio council, I want to see all the activities that take up people's time, who is involved in what, and whether these activities currently need our attention or not (Overview)**
>
> **As a member of the portfolio council, I want to be able to adjust the allocation of resources (i.e. who is assigned to what) for the timeframe until the next portfolio review to make sure that what from the business perspective is important gets attended to (Resource allocation)**

For portfolio review decision-making (see Section 8.2 on *Setting up agile-compatible portfolio management*), it is crucial to see the relative importance of the activities as well as their cadence. This yields the following user stories:

> **As a member of the portfolio council, I want to see when the activities' upcoming control points (e.g. planning meetings and demos) occur in order to better understand their current relative importance (Cadence overview)**
>
> **As a member of the portfolio council, I want to see and be able to set the relative importance of the ongoing activities until the next portfolio review in order to guide decision-making during that timeframe (Activity ranking)**

It is also important for the portfolio council to easily see more details concerning the activities themselves when necessary:

> **As a member of the portfolio council, I want to be able to quickly see details concerning an individual activity and possibly "drill down" into it in order to better understand its actual status (Drill)**

Agilefant's portfolio overview supports at least to some degree the *Overview*, *Resource allocation* and *Activity ranking* user stories described above. It is displayed in Figure 13.6 below:

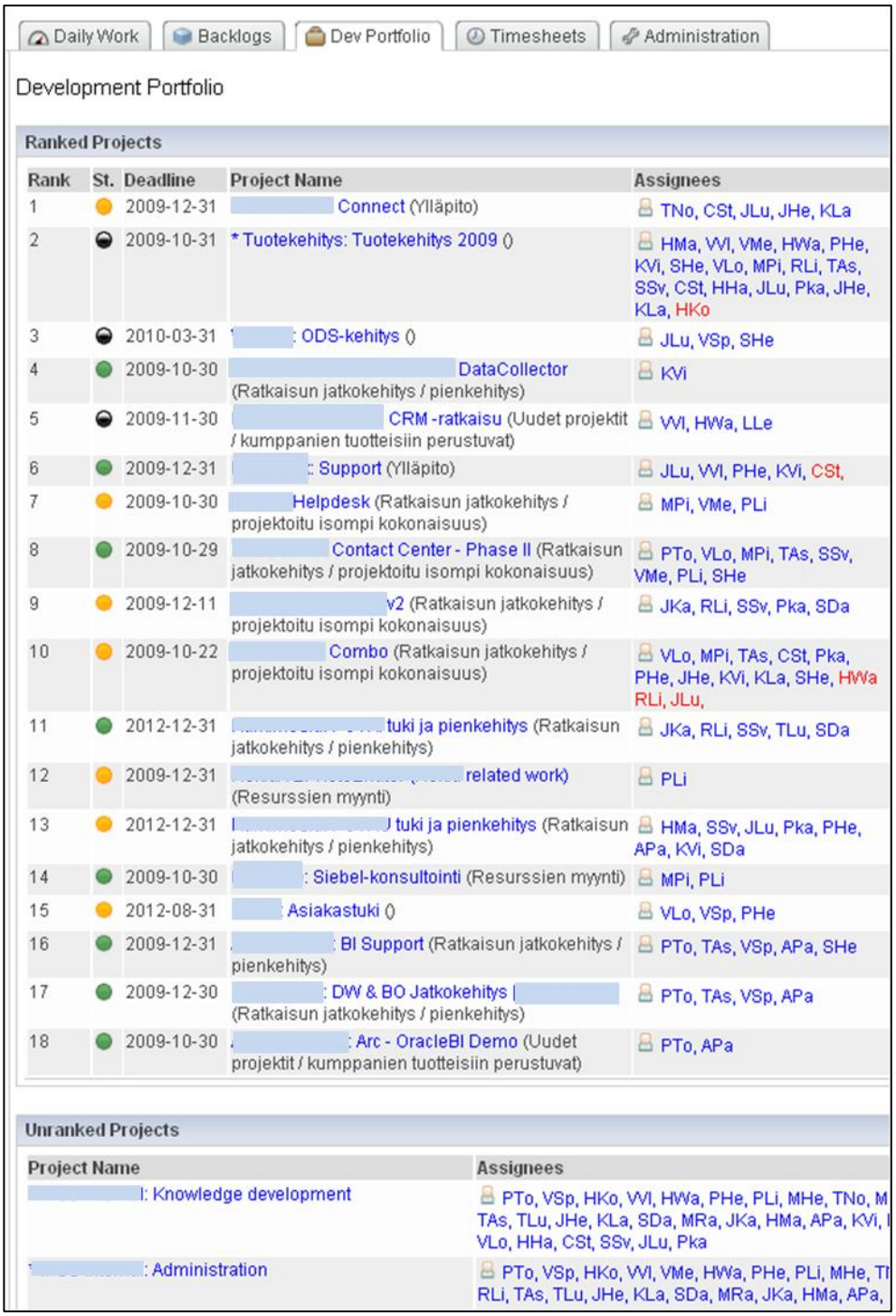


Figure 13.6: Portfolio overview in Agilefant 2.0.4

The development portfolio overview shown in Figure 13.6 displays the portfolio of activities currently ongoing in a case company of some 30 people and the ranking of the activities' relative importance. While the set priority is not absolute in the sense that all of the work in the activities ranked as most important would be more important than all of the work in the activities that have been ranked as less important, this explicit priority helps in making trade-offs when necessary. Also, performing the ranking forces the Portfolio Council to keep

things explicit, and thus in its own way serves to communicate the strategy of the company. The traffic lights denote the status of the activities as judged by their owners. In this example, the green color indicates that an activity is proceeding as expected. Black denotes that an activity is not managed in detail using Agilefant, and thus, its status cannot be deducted from the system. In 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 can still be represented in Agilefant. The list of 'unranked projects' continues beyond the bottom edge of the picture, with the total number of ongoing activities being around 40. The name of the organization itself (as well as the names of its clients have been blanked out from Figure 13.6 for confidentiality reasons.

Besides the user stories listed above, it is relatively easy to invent requirements for a "perfect" portfolio overview. Thus, there is little point to list them all here in the user story format. To shed some additional light on what a good portfolio overview might look like, Figure 13.7 and Figure 13.8 below present a couple of prototypes we have come up with during the past years. Table 13.1 presents a legend of the notation used in the first of these visualizations (Figure 13.7), while Figure 13.8 has been built using the more self-explanatory user interface components currently provided in Agilefant. You can also compare Figure 13.7 to Figure 3.6 in Part I, as both contain the same activities.

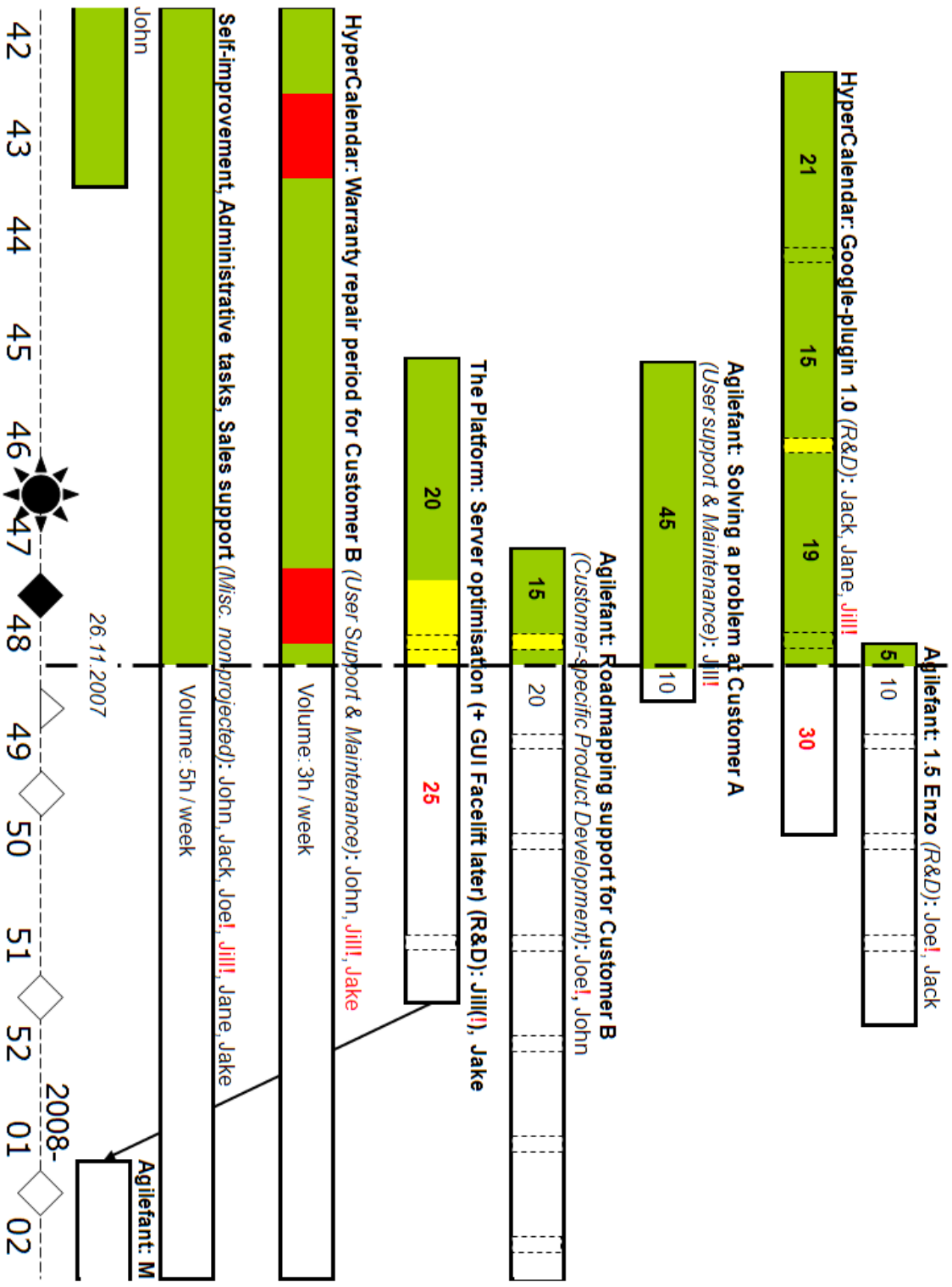
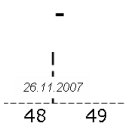
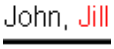
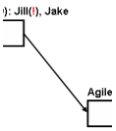


Figure 13.7: An example portfolio overview visualization

Table 13.1: Legend for the notation used in the example portfolio visualization (Figure 13.7)

Graphic	Explanation
	**The scaling** of the visualization automatically displays centers on the current date, and displays six weeks forwards and backwards. The visualization is scrollable. **The timeline** – a dashed horizontal line at the bottom of the visualization – displays weeks and the turn of the year. The **current date** is denoted with a horizontal dashed line.
	**Development activities** are displayed as *horizontal bars* of different colors (see explanation of Status below) In addition to the currently ongoing development activities, those that have ended or start during the period displayed are shown. Activities on hold, waiting to be launched, or being prepared for in sales are omitted. Non-projected development activities (e.g. support activities) are shown as bars stretching across the entire visualization. **Product**, **project** and **activity type names** are displayed above each activity.
	**Activity status** can be green, yellow or red. Green means that the team's best guess is that all planned iteration goals (or project goals, in the case of non-iterative activities) are going to be met. Yellow means that the team considers it possible that at least one of the iteration goals is in jeopardy but immediate attention from the Business is not needed as they may be able to resolve the situation on their own. Red means that the team believes that at least one of the goals will not be met, or when immediate attention from Business is needed.
	**Iteration review** meeting at the end of an iteration. While Agilefant automatically takes on the current status of the iteration to represent whether iteration goals were met or not, it is possible to change the status of an iteration review at any time. This is because an iteration review may reveal that the goals were not in met even though the activity proceeded as *green*.
	**The historical velocity** of the iteration (or an activity, in case of non-iterative activities) is calculated as the sum of the *original effort estimates* of the backlog items 'Done' in the iteration.
	**Estimated effort left** for the rest of the iteration (or an activity, in case of non-iterative activities) is calculated as the sum of the estimated efforts left for the backlog items not in the 'Done' state. It is displayed in red if the realized velocity for the activity so far is not enough to get the effort left done on time. In the case of non-projected development activities, its current volume (i.e. the average amount of weekly attention per involved person is required by the activity) is shown. Currently, historical volume is not tracked.
	Priorities for ongoing activities are shown in their vertical placing in the visualization, from the most important (at the top) to the least important (at the bottom). Activities that are not ongoing but start or end within the displayed period are displayed at the bottom of the visualization, below the least important ongoing activity.
	**The people involved** in the ongoing iteration for each activity are displayed on top of the activity after the names of the product, project and activity type. Those people who have tasks or backlog items assigned but have not been appointed to the activity by Development Portfolio Management are displayed in red.
	**Personal overload** occurs when a person has more estimated effort left in terms of backlog items and tasks assigned to him than he, compared to his personal historical velocity, is likely to be able to get done before his first upcoming iteration deadlines. Personal overload is denoted with a red exclamation mark after the name of the person.
	**Dependencies** in terms of activities' contents are denoted with an arrow. There is no extra notation to denote resource dependencies, as they in principle can be seen from assignments and personal overload (see above)
	**Portfolio control points** (see section **Error! Reference source not found.**on p. 143.are shown on the timeline as a *sun* (strategy day & roadmap revision), a *diamond* (portfolio review) or a *triangle* (traffic control meeting).

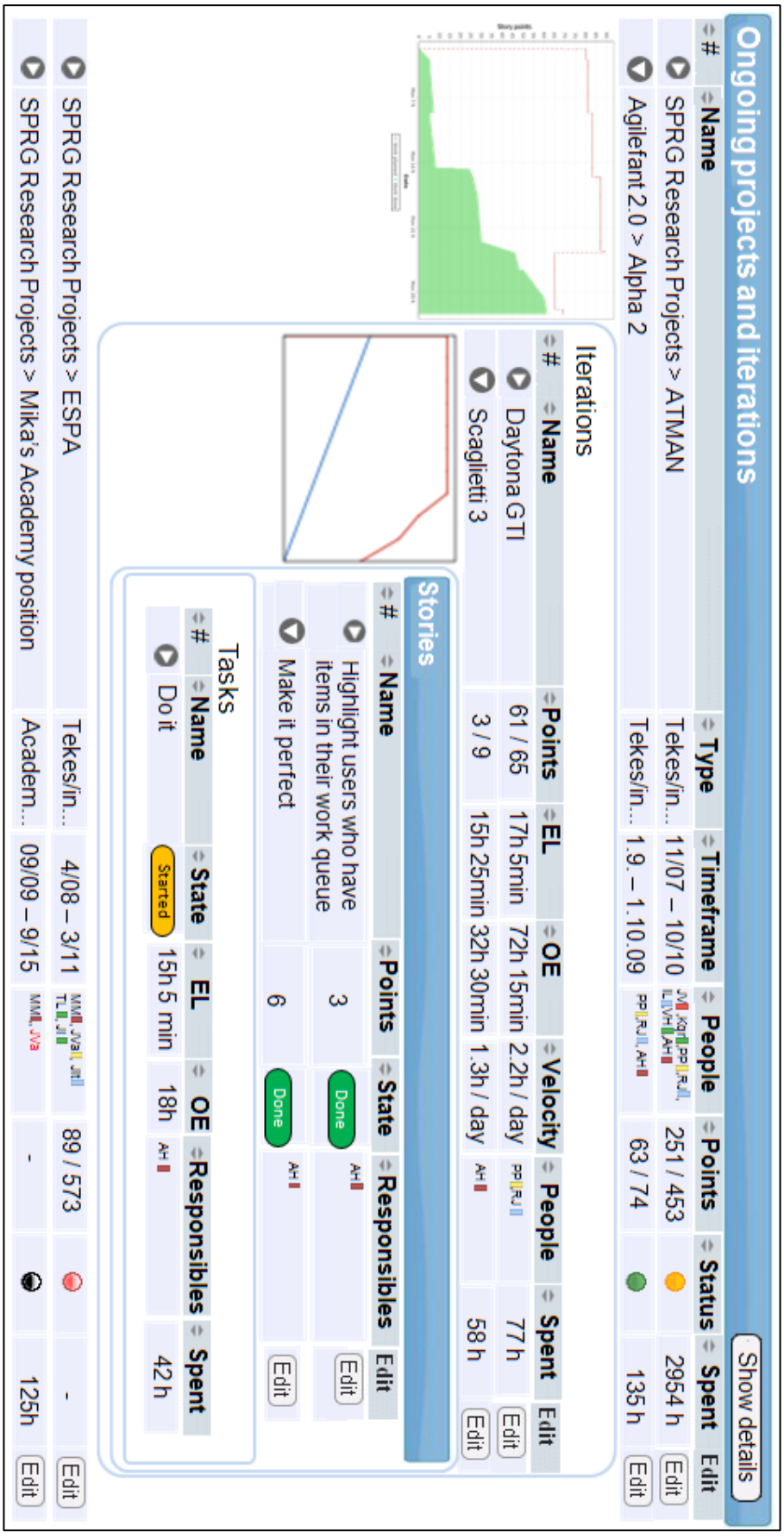


Figure 13.8: Another prototype portfolio overview from Agilefant

Finally, while the portfolio overview may be the "basic" view utilized in portfolio reviews, some people are usually interested in having customized views into the development portfolio that provide more details of the activities (this could also be iterations, people, individual stories, etc.) they are interested in while leaving out the rest:

> **As (e.g.) a product manager, I want to be able to see at a glance how the three concurrent development iterations that feed into my product are progressing, as well as whether my key product owner is overloaded or not (Customized views)**

These kinds of customized views may also be the more feasible way of providing – or at least implementing – all the information presented in the portfolio overviews presented in the above prototypes.

13.3.2 Load management

To properly manage the development portfolio, you need to account for all of the work that takes up time from people. This includes both projects as well as non-project activities:

> **As the scrum master, in addition to project work, I need to be able to account for non-project work in order to stay aware of my team's workload and all that needs to be taken care of (Continuous work)**

Like explained in Section 8.1.5 (*Time management conducted by individuals*), continuous work is best to be carried out as periodic work whenever possible. As a simple example, in one of our case companies, a product manager always reserved 30 minutes at the end of each day to take care of incoming email. Thus, he was able to refrain from constantly interrupting himself by checking his email.

> **As a team member responsible for (a part of) some continuous work, e.g. responding to support requests, I want the backlog management system to support me in transforming continuous work into periodic work in order to stay efficient and refrain from unnecessary multi-tasking (Periodic work)**

Also, projects can contain continuous work that takes time from people and needs to be accounted for, but may not be worth the effort to explicitly transform it into periodic work and/or model it as work items or tasks. Examples of such work are 5% workshops with the product owner, sprint and iteration planning meetings, and daily meetings. This yields the following user story:

> **As the scrum master, I want my team to account in their workload for the different continuous work they are attending to**

> **in order to keep both themselves as well as the product owner aware of the team's true capacity (Baseline load [37])**

Also, if wished, slack and/or overhead (as these are bound to occur anyhow) could be accounted for in the baseline load.

13.4 Daily work

As explained in Section 8.1.5 (*Time management conducted by individuals*) on page 131, it is ultimately up to the individual to make sure he uses the time he has available efficiently. But this can be supported by proper backlog management. This yields the following user story:

> **As a team (or an individual) I want to collect all of my work from the ongoing activities summarized in a single view to help me choose what I should take on next, and help the product owners competing for my attention to agree on priorities. (Floating backlog [38])**

The concept of floating backlogs is explained in-depth in Section 9.2 (Controlled multi-tasking with floating backlogs) on page 149.

While Agilefant 2.0.4 does not currently support floating backlogs, it has a relatively sophisticated Daily work view that individuals can use for load management as well as planning their daily work. This is displayed in Figure 13.9 below:

[37] Agilefant 2.0.4's Daily work view handles this from the perspective of an individual, but not a team. Also, the work items in the current implementation are not displayed in any particular order, and they cannot be re-prioritized from the view.

[38] While Agilefant 2.0.4 does not implement continuous work, you can work around this by creating projects that span a long time and adding baseline load to them.

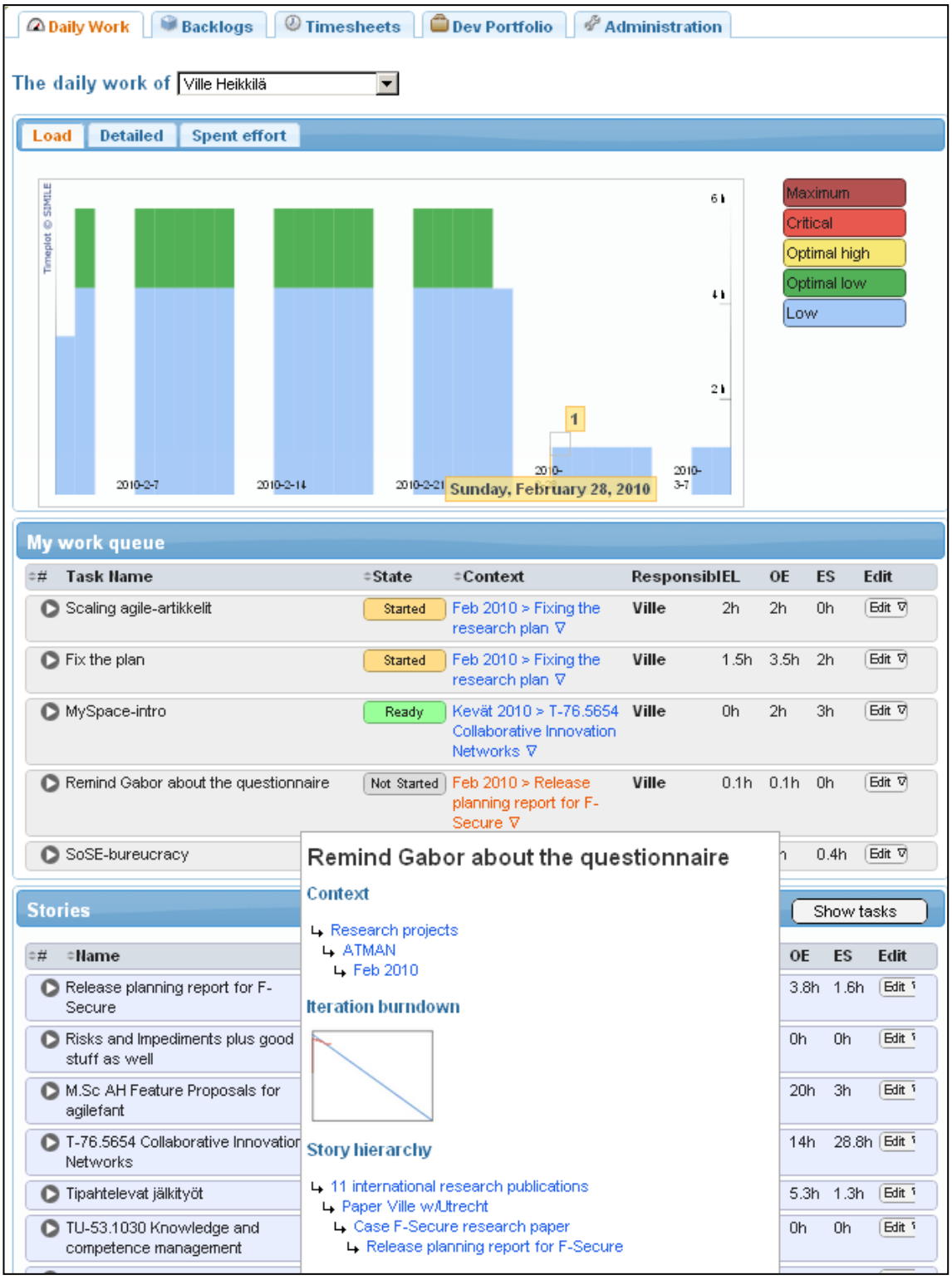


Figure 13.9: The daily work view in Agilefant 2.0.4 summarizes and helps organize the duties of an individual across the activities he is involved in

The daily work view (Figure 13.9) is designed to help individuals deal with the "bottom level of portfolio decision-making" as explained in Section 8.1 (*Levels of portfolio management in an agile enterprise*). It summarizes the duties of an individual from all of the activities he is currently involved in, draws a graph of his current and upcoming workload, and helps in selecting and organizing the

task(s) to be attended to next in the bustle of multiple roles and responsibilities that pervades at least in small organizations developing software.

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 being assigned as one of the people responsible for a given task) or indirectly (via being assigned to an iteration containing unassigned tasks) responsible for, and compares the total against set thresholds. The middle section of Figure 13.9 displays the person's *work queue*, which is a personal space 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 activities' and the stories' relative priorities. However, there is always room for intuition in picking the "next most relevant task" depending on e.g. the mood of the individual, or 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 in 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 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 13.9 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.

As explained in Chapter 9*: Agile Development Portfolio Management* in the hypothetical "pure 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. However, it seems that the vast majority of organizations are *in the process* of adopting agile software development for *some* of their activities only, rather than practicing "pure Scrum" for everything. Also, as explained, there virtually always are multiple ongoing activities that require the development resources attention. Thus, it is plausible that the daily work view is relevant in practice.

A complete list of potential requirements for supporting the management of daily work gained from surveying a number of expert practitioners is presented in Appendix 3 of Haapala (2010).

REFERENCES

Abrahamsson, P. 2003, "New directions on agile methods: A comparative analysis", in *Proceedings of the International Conference on Software Engineering (ICSE 2003),* Portland, Oregon: IEEE Computer Society, pp. 244-254.

Akker, M.,van den, Brinkkemper, S., Diepen, G. & Versendaal, J. 2008, "Software product release planning through optimization and what-if analysis", *Information and Software Technology,* vol. 50, no. 1-2, pp. 101-111.

Al-Emran, A., Pfahl, D. & Ruhe, G. 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.

Anavi-Isakow, S. & Golany, B. 2003, "Managing multi-project environments through constant work-in-process", *International Journal of Project Management,* vol. 21, no. 1, pp. 9-18.

Anderson, D.,J. 2010, *Kanban,* Sequim, Washington: Blue Hole Press.

Artz, P., Weerd, I.,van de, Brinkkemper, S. & Fieggen, J. 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.

Ash, R.C. & Smith-Daniels, D.E. 2004, "Managing the Impact of Customer Support Disruptions on New Product Development Projects", *Project Management Journal,* vol. 35, no. 1, pp. 3-10.

Basili, V.R., Caldiera, G. & Rombach, D.,H. 1994, "The Experience Factory" in *Encyclopaedia of Software Engineering*, ed. J.J. Marciniak, John Wiley & Sons, pp. 469-476.

Beattie, J.S. & Fleck, J. 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, K. 2000, *Extreme Programming Explained: Embrace Change,* Reading, MA: Addison-Wesley.

Bekkers, W., Weerd, I.,van de, Spruit, M. & Brinkkemper, S. 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, P. & Jönsson, P.R. 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.

Berry, M. 1998, "Strategic planning in small high-tech companies", *Long range planning,* vol. 31, no. 3, pp. 455-466.

Berry, M.M.J. & Taggart, J.H. 1998, "Combining Technology and Corporate Strategy in Small High Tech Firms", *Research Policy,* vol. 26, no. 7-8, pp. 883-895.

Blichfeldt, B.S. & Eskerod, P. 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, B.W. 2000, "Requirements that handle IKIWISI, COTS, and rapid change", *Computer,* vol. 33, no. 7, pp. 99-102.

Boehm, B.W. & Turner, R. 2003, *Balancing agility and discipline : a guide for the perplexed,* Boston, MA: Addison-Wesley.

Booch, G. 1995, *Object Solutions: Managing the Object-Oriented Project,* Boston, MA: Addison-Wesley.

Brooks, F.P.,Jr. 1995, *The Mythical Man-Month: Essays on Software Engineering,* 20th anniversary ed., Reading, MA, USA: Addison-Wesley.

Brown, S.,L. & Eisenhardt, K.,M. 1997, "The Art of Continuous Change: Linking Complexity Theory and Time-Paced Evolution in Relentlessly Shifting Organizations", *Administrative Science Quarterly,* vol. 42, no. 1, pp. 1-34.

Cao, L. & Ramesh, B. 2008, "Agile requirements engineering practices: An empirical study", *IEEE Software,* vol. 25, no. 1, pp. 60-67.

Cardozo, E.S.F., Neto, J.B.F.A., Barza, A., Franca, A.C.C. & Da Silva, F.Q.B. 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.

Carlshamre, P. 2002, "Release Planning in Market-Driven Software Product Development: Provoking an Understanding", *Requirements Engineering,* vol. 7, no. 3, pp. 139-151.

Cerveny, J.F. & Galup, S.D. 2002, "Critical chain project management holistic solution aligning quantitative and qualitative project management methods", *Production and Inventory Management Journal,* vol. 43, no. 3-4, pp. 55-64.

Chatzipetrou, P., Angelis, L., Rovegård, P. & Wohlin, C. 2010, "Prioritization of Issues and Requirements by Cumulative Voting: A Compositional Data Analysis Framework", in *Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications,* Lille, France: IEEE Computer Society, pp. 361-370.

Chow, T. & Cao, D. 2008, "A survey study of critical success factors in agile software projects", *Journal of Systems and Software,* vol. 81, no. 6, pp. 961-971.

Christensen, C.M. & Raynor, M.E. 2003, *The Innovator's Solution: Creating and Sustaining Successful Growth,* Boston, MA: Harvard Business School Press.

Cockburn, A. 2002, *Agile software development,* Boston, MA: Addison-Wesley.

Cohn, M. 2010, *Succeeding with Agile: Software Development Using Scrum,* Upper Saddle River, NJ: Addison-Wesley.

Condon, D. 2002, *Software product management - managing software development from idea to product to marketing to sales,* Boston, MA: Aspatore Books.

Cooper, R.G. 2009, "How Companies are Reinventing Their Idea-to-Launch Methodologies", *Research Technology Management,* vol. 52, no. 2, pp. 47-57.

Cooper, R.G. 1994, "Perspective Third-Generation New Product Processes", *Journal of Product Innovation Management,* vol. 11, no. 1, pp. 3-14.

Cooper, R.G., Edgett, S.J. & Kleinschmidt, E.J. 2001, *Portfolio Management for New Products,* 2nd ed., Cambridge, MA: Perseus Books.

Cooper, R.G. & Edgett, S.J. 2003, "Overcoming the crunch in resources for new product development", *Research Technology Management,* vol. 46, no. 3, pp. 48-58.

Cooper, R.G., Edgett, S.J. & Kleinschmidt, E.J. 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.

Cooper, R.G., Edgett, S.J. & Kleinschmidt, E.J. 2001, "Portfolio management for new product development: results of an industry practices study", *R&D Management,* vol. 31, no. 4, pp. 361-380.

Cooper, R.G., Edgett, S.J. & Kleinschmidt, E.J. 2000, "New problems, new solutions: Making portfolio management more effective", *Research Technology Management,* vol. 43, no. 2, pp. 18-33.

Cooper, R.G., Edgett, S.J. & Kleinschmidt, E.J. 1997, "Portfolio management in new product development: Lessons from the leaders--II", *Research Technology Management,* vol. 40, no. 6, pp. 43-52.

Cusumano, M.A. 2008, "The Changing Software Business: Moving from Products to Services", *Computer,* vol. 41, no. 1, pp. 20-27.

Cusumano, M.A. 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, M.A. 2003, "Finding your balance in the products and services debate", *Communications of the ACM,* vol. 46, no. 3, pp. 15-17.

Cusumano, M.A., Crandall, W., MacCormack, A. & Kemerer, C.,F. 2009, "Critical Decisions in Software Development: Updating the State of the Practice", *IEEE Software,* vol. 26, no. 5, pp. 84-87.

Cusumano, M.A. & Selby, R.W. 1995, *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets and Manages People,* New York: The Free Press.

De Reyck, B., Grushka-Cockayne, Y., Lockett, M., Calderini, S.R., Moura, M. & Sloper, A. 2005, "The impact of project portfolio management on information technology projects", *International Journal of Project Management,* vol. 23, no. 7, pp. 524-537.

DeGregorio, G. 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.

DeMarco, T. 2001, *Slack: Getting Past Burnout, Busywork, and the Myth of Total Efficiency,* New York, NY: Broadway Books.

Dobson, M.S. 1999, *The juggler's guide to managing multiple projects,* Newtown Square, PA: Project Management Institute.

Dooley, L., Lupton, G. & O'Sullivan, D. 2005, "Multiple project management: a modern competitive necessity", *Journal of Manufacturing Technology Management,* vol. 16, no. 5, pp. 466-482.

Doz, Y. & Kosonen, M. 2008, *Fast Strategy: How strategic agility will help you stay ahead of the game,* Upper Saddle River, NJ: Prentice Hall.

Dybå, T. & Dingsøyr, T. 2008, "Empirical studies of agile software development: A systematic review", *Information and Software Technology,* vol. 50, no. 9-10, pp. 833-859.

Ebert, C. 2009, "Software Product Management", *Crosstalk, The Journal of Defence Software Development,* vol. 2009, no. January, pp. 15-19.

Eisenhardt, K.,M. & Brown, S.,L. 1998, "Time Pacing: Competing in Markets that Won't Stand Still", *Harvard business review,* vol. 76, no. 2, pp. 59-69.

Elonen, S. & Artto, K.A. 2003, "Problems in managing internal development projects in multi-project environments", *International Journal of Project Management,* vol. 21, no. 6, pp. 395-402.

Englund, R.L. & Graham, R.J. 2001, "From experience: linking projects to strategy", *IEEE Engineering Management Review,* vol. 28, no. 1, pp. 58-69.

Engwall, M. & Jerbrant, A. 2003, "The resource allocation syndrome: The prime challenge of multi-project management?", *International Journal of Project Management,* vol. 21, no. 6, pp. 403-409.

Ferrari, E. 2008, *Product Management for Software - simple processes for great results,* Mondo Strategies Press.

Fisher, K.G. & Bankston, A. 2009, "From Cradle to Sprint: Creating a Full-Lifecycle Request Pipeline at Nationwide Insurance", in *Proceedings of the 2009 Agile Conference (Agile2009),* Chicago, IL: IEEE computer society, pp. 223-228.

Fleury, A.L., Hunt, F., Spinola, M. & Probert, D. 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.

Fogelström, N.D., Gorschek, T., Svahnberg, M. & Olsson, P. 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.

Fricker, S., Gorschek, T., Byman, C. & Schmidle, A. 2010, "Handshaking with Implementation Proposals: Negotiating Requirements Understanding", *IEEE Software,* vol. 27, no. 2, pp. 72-80.

Galen, R. 2009, *Scrum product ownership: balancing value from the inside out - stories, ideas, lessons and practices for becoming a great product owner,* RGalen Consulting Group.

Gersick, C.J.G. 1994, "Pacing Strategic Change: The Case of a New Venture", *Academy of management journal,* vol. 37, no. 1, pp. 9-45.

Glass, R.L., Vessey, I. & Ramesh, V. 2002, "Research in software engineering: an analysis of the literature", *Information & Software Technology,* vol. 44, no. 8, pp. 491-506.

Gorchels, L. 2003, *The product manager's field guide,* New York, NY: McGraw-Hill.

Gorschek, T. & Wohlin, C. 2006, "Requirements Abstraction Model", *Requirements Engineering,* vol. 11, no. 1, pp. 79-101.

Greer, D. & Ruhe, G. 2004, "Software Release Planning: An Evolutionary and Iterative Approach", *Information and Software Technology,* vol. 46, no. 4, pp. 243-253.

Haapala, A. 2010, *Enhanced tool support for daily work management in agile software development*, master's thesis, Oulu University, industrial engineering and management.

Hansson, C., Dittrich, Y., Gustafsson, B. & Zarnak, S. 2006, "How agile are industrial software development practices?", *Journal of Systems and Software,* vol. 79, no. 9, pp. 1295-1311.

Harris, J.R. & McKay, J.C. 1996, "Optimizing product development through pipeline management" in *The PDMA handbook of new product development*, ed. M.D. Rosenau Jr., New York, NY: John Wiley & Sons, pp. 63-76.

Heikkilä, V., Jadallah, A., Rautiainen, K. & Guenther, R. 2010, "Rigorous support for flexible planning of product releases - a stakeholder-centric approach", in *Proceedings of the 43th Hawaii International Conference on System Sciences (HICSS-43),* Kauai, HI: IEEE, pp. 1-10.

Heikkilä, V., Rautiainen, K. & Jansen, S. 2010, "A Revelatory Case Study on Scaling Agile Release Planning", in *Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO SEAA 2010),* Lille, France: IEEE, pp. 289-296.

Highsmith, J.A.,III 2002, *Agile software development ecosystems,* Boston, MA: Addison-Wesley.

Highsmith, J.A.,III 2000, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems,* New York, NY: Dorset House Publishing.

Hodgkins, P. & Hohmann, L. 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.

Ivarsson, M. & Gorschek, T. 2009, "Technology transfer decision support in requirements engineering: a systematic review of REj", *Requirements Engineering,* vol. 14, no. 3, pp. 155-175.

Jennings, P. & Beaver, G. 1996, "The Performance and Competitive Advantage of Small Firms: A Management Perspective", *International Small Business Journal,* vol. 15, no. 2.

Johnson, M.W. 2010, *Seizing the white space: business model innovation for growth and renewal,* Boston, MA: Harvard Business Press.

Kahn, K.B., Castellion, G.A. & Griffin, A. (eds) 2005, *The PDMA handbook of new product development,* 2nd ed., Hoboken, NJ: John Wiley & Sons.

Kappel, T. 2001, "Perspectives on roadmaps: how organisations talk about the future", *IEEE Engineering Management Review,* vol. 29, no. 3, pp. 36-48.

Karlsson, J., Olsson, S. & Ryan, K. 1997, "Improved practical support for large-scale requirements prioritising", *Requirements Engineering,* vol. 2, no. 1, pp. 51-60.

Karlsson, J. & Ryan, K. 1997, "A cost-value approach for prioritizing requirements", *IEEE Software,* vol. 14, no. 5, pp. 67-74.

Karlström, D. & Runeson, P. 2006, "Integrating agile software development into stage-gate managed product development", *Empirical Software Engineering,* vol. 11, no. 2, pp. 203-225.

Kaulio, M.A. 2008, "Project leadership in multi-project settings: Findings from a critical incident study", *International Journal of Project Management,* vol. 26, no. 4, pp. 338-347.

Kettunen, P. 2007, "Extending Software Project Agility with New Product Development Enterprise Agility", *Software Process Improvement and Practice,* vol. 12, no. 6, pp. 541-548.

Kettunen, P. & Laanti, M. 2006, "Combining agile software projects and large-scale organizational agility", *Software Process Improvement and Practice,* vol. 13, no. 2, pp. 183-193.

Kittlaus, H. & Clough, P.N. 2009, *Software product management and pricing: key success factors for software organizations,* Berlin, Germany: Springer-Verlag.

Kniberg, H. & Skarin, M. 2010, *Kanban and Scrum - making the most of both,* lulu.com.

Koivisto, N. 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, R.N. & Schaller, R.R. 2001, "Science and Technology Roadmaps", *IEEE Transactions on Engineering Management,* vol. 48, no. 2, pp. 132-143.

Krebs, J. 2008, *Agile Portfolio Management,* Redmond, WA: Microsoft Press.

Kruchten, P. 2000, *The rational unified process: an introduction,* 2nd ed., Reading, MA: Addison-Wesley.

Ktata, O. & Levesque, G. 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.

Ladas, C. 2008, *Scrumban,* Seattle, Washington: Modus Cooperandi Press.

Lago, P., Muccini, H. & Vliet, H.,van 2009, "A scoped approach to traceability management", *Journal of Systems and Software,* vol. 82, no. 1, pp. 168-182.

Larman, C. & Vodde, B. 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, C. & Vodde, B. 2008, *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum,* Westford, MA: Addison-Wesley Professional.

Laslo, Z. & Goldberg, A.I. 2008, "Resource allocation under uncertainty in a multi-project matrix environment: Is organizational conflict inevitable?", *International Journal of Project Management,* vol. 26, no. 8, pp. 773-788.

Leffingwell, D. forthcoming 2011, *Agile Requirements: Lean Requirements Practices for Software Teams, Programs and the Enterprise,* Addison-Wesley Professional.

Leffingwell, D. 2009, *The Big Picture of Enterprise Agility (rev. 2)*, http://scalingsoftwareagility.files.wordpress.com/2007/03/the-big-picture-of-enteprise-agilitywhitepaper.pdf.

Leffingwell, D. 2007, *Scaling Software Agility: Best Practices for Large Enterprises,* Reading, MA: Addison-Wesley Professional.

Leffingwell, D. & Aalto, J. 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, I. 2010, *Using backlogs for linking long-term product plans and development tasks in agile software development*, master's thesis, Helsinki University of Technology, Software Business and Engineering Institute.

Lehto, I. & Rautiainen, K. 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, L. 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.

Lehtola, L. & Kauppinen, M. 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, L., Kauppinen, M., Komssi, M. & Vähäniitty, J. 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, L., Kauppinen, M. & Kujala, S. 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.

Lehtola, L., Kauppinen, M. & Vähäniitty, J. 2007, "Strengthening the link between business decisions and RE: Long-term product planning in software product companies", in *Proceedings of the 15th IEEE International Requirements Engineering Conference (RE'07),* New Delhi, India: IEEE, pp. 153-162.

Levine, H.A. 2005, *Project Portfolio Management: A Practical Guide to Selecting Projects, Managing Portfolios, and Maximizing Benefits,* San Francisco, CA: Jossey-Bass.

Li, M., Huang, M., Shu, F. & Li, J. 2006, "A risk-driven method for eXtreme programming release planning", in *Proceedings of the 28th International Conference on Software Engineering (ICSE 2006),* Shanghai, China: ACM, pp. 423-430.

MacCormack, A., Verganti, R. & Iansiti, M. 2001, "Developing products on Internet time: the anatomy of a flexible development process", *Engineering management review,* vol. 29, no. 2, pp. 90-104.

Mader, D.P. 2004, "Selecting Design for Six Sigma Projects", *Quality Progress,* vol. 37, no. 7, pp. 65.

Maham, M. 2008, "Planning and Facilitating Release Retrospectives", in *Proceedings of the Agile 2008 conference (AGILE '08),* Toronto, Canada: IEEE, pp. 176-180.

Martin, J. 1991, *Rapid Application Development,* New York, NY: Macmillan Publishing Company.

Martinsuo, M. 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.

Martinsuo, M. & Lehtonen, P. 2007, "Role of single-project management in achieving portfolio management efficiency", *International Journal of Project Management,* vol. 25, no. 1, pp. 56-65.

Mc Elroy, J. & Ruhe, G. 2010, "When-to-release decisions for features with time-dependent value functions", *Requirements Engineering,* vol. 15, no. 3, pp. 337-358.

McCarthy, J. & McCarthy, M. 2002, *Software for your head - core protocols for creating and maintaining shared vision,* Pearson Education, Inc.

McConnell, S. 1996, *Rapid Development,* Redmond, WA: Microsoft Press.

McDonough, E.F. & Spital, F.C. 2003, "Managing project portfolios", *Research Technology Management,* vol. 46, no. 3, pp. 40-46.

McGrath, M. 2000, *Product Strategy for High Technology Companies,* McGraw Hill Text.

McGrath, M.E. (ed) 1996, *Setting the PACE in product development : a guide to Product And Cycle-time Excellence,* Boston, MA: Butterworth-Heinemann.

Miettinen, O., Mazhelis, O. & Luoma, E. 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.

Mohan, K., Ramesh, B. & Sugumaran, V. 2010, "Integrating Software Product Line Engineering and Agile Development Methods", *IEEE Software,* vol. 23, no. 3, pp. 48-55.

Nambisan, S. 2001, "Why Service Businesses are not Product Businesses", *MIT Sloan Management Review,* vol. 42, no. 4, pp. 72-80.

Nambisan, S. & Wilemon, D. 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, A. & Ruhe, G. 2009, "Optimized Resource Allocation for Software Release Planning", *IEEE Transactions on Software Engineering,* vol. 35, no. 1, pp. 109-123.

Ngo-The, A. & Ruhe, G. 2008, "A systematic approach for solving the wicked problem of software release planning", *Soft Computing - A Fusion of Foundations, Methodologies and Applications,* vol. 12, no. 1, pp. 95-108.

O'Connor, P. 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.

Patton, M.Q. 2002, *Qualitative Research & Evaluation Methods,* 3rd ed., Thousand Oaks, CA: Sage Publications.

Payne, J.H. 1995, "Management of multiple simultaneous projects: A state-of-the-art review", *International Journal of Project Management,* vol. 13, no. 3, pp. 163-168.

Pichler, M., Rumetshofer, H. & Wahler, W. 2006, "Agile Requirements Engineering for a Social Insurance for Occupational Risks Organization: A Case Study", in *Proceedings of the 14th IEEE International International Requirements Engineering Conference (RE'06),* Minneapolis-St. Paul, MN: IEEE, pp. 251-256.

Pichler, R. 2010, *Agile Product Management with Scrum: Creating Products that Customers Love,* Addison-Wesley Professional.

Pino, F.J., Pardo, C., García, F. & Piattini, M. 2010, "Assessment methodology for software process improvement in small organizations", *Information and Software Technology,* vol. 52, no. 10, pp. 1044-1061.

Pittman, M. 1993, "Lessons Learned in Managing Object-Oriented Development", *IEEE Software,* vol. 10, no. 1, pp. 43-53.

Poppendieck, M. & Poppendieck, T. 2009, *Leading Lean Software Development: Results Are not the Point,* Addison-Wesley.

Ramesh, B., Cao, L. & Baskerville, R. 2010, "Agile requirements engineering practices and challenges: an empirical study", *Information Systems Journal,* vol. 20, no. 5, pp. 449-480.

Rautiainen, K. 2004, *Cycles of Control: A Temporal Pacing Framework for Software Product Development Management*, licentiate's thesis , Helsinki University of Technology.

Rautiainen, K., Lassenius, C. & Sulonen, R. 2002, "4CC: A framework for managing software product development", *EMJ - Engineering Management Journal,* vol. 14, no. 2, pp. 27-32.

Rautiainen, K., Lassenius, C., Vähäniitty, J., Itkonen, J., Mäntylä, M., Rusama, M. & Vanhanen, J. 2006, *Pacing Software Product Development: A Framework and Practical Implementation Guidelines* , Helsinki University of Technology, Software Business and Engineering Institute, Espoo, Finland.

Reinertsen, D.G. 2009, *The principles of product development flow - second generation lean product development,* Redondo Beach, CA: Celeritas publishing.

Repenning, N.P. 2001, "Understanding fire fighting in new product development", *The Journal of Product Innovation Management,* vol. 18, no. 5, pp. 285-300.

Rico, D.F., Sayani, H.H. & Sone, S. 2009, *The business value of agile software methods - maximizing ROI with just-in-time processes and documentation,* J. Ross Publishing.

Rönkkö, M., Ylitalo, J., Peltonen, J., Koivisto, N., Mutanen, O., Autere, J., Valtakoski, A. & Pentikäinen, P. 2009, *National Software Industry Survey 2009*, Helsinki University of Technology.

Rothman, J. 2009, *Manage your project portfolio: increase your capacity and finish more projects,* Raleigh, NC: Pragmatic Bookshelf.

Rothman, J. 2007, *Manage it! : your guide to modern, pragmatic project management,* Raleigh, NC: The Pragmatic Bookshelf.

Ruhe, G. 2010, *Product Release Planning : Methods, Tools and Applications*, Boca Raton, FL: Auerbach Publications.

Ruhe, G. & Momoh, J. 2005, "Strategic Release Planning and Evaluation of Operational Feasibility", in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS '05),* Hawaii, HI: IEEE, pp. 313b.

Ruokonen, M. 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, I. & Tukiainen, M. 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.

Sahlman, K. & Haapasalo, H. 2009, "Perceptions of strategic management of technology in small high-tech enterprises", in *Proceedings of the 2009 Portland International Conference on Management of Engineering & Technology (PICMET 2009),* Portland, OR: IEEE, pp. 93-104.

Savolainen, J., Kuusela, J. & Vilavaara, A. 2010, "Transition to agile development - rediscovery of important requirements engineering practices", in *Proceedings of the 18th International IEEE Requirements Engineering Conference (RE'10),* Sydney, Australia: IEEE Computer Society, pp. 289-294.

Schiel, J. 2009, *Enterprise-Scale Agile Software Development,* Boca Raton: CRC Press.

Schwaber, K. 2007, *The Enterprise and Scrum,* Redmond, WA: Microsoft Press.

Schwaber, K. 1995, "SCRUM Development Process", in *Proceedings of the OOPSLA Workshop on Business Object Design and Implementation,* Austin, TX: pp. 117-134.

Schwaber, K. & Beedle, M. 2002, *Agile software development with Scrum,* Upper Saddle River, NJ: Prentice-Hall.

Schwaber, K. & Sutherland, J. 2010, *Scrum Guide*, Agile Alliance, http://www.scrum.org/storage/scrumguides/Scrum%20Guide.pdf.

Shalloway, A., Beaver, G. & Trott, J. 2009, *Lean-agile software development: achieving enterprise agility,* Upper Saddle River, NJ: Addison-Wesley.

Smith, P.G. 2008, "Change: embrace it, don't deny it", *Research-Technology Management,* vol. 51, no. 4, pp. 34-40.

Stober, T. & Hansmann, U. 2010, *Agile software development - best practices for large software development projects,* Springer.

Sutherland, J. & Altman, I. 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, J. 2005, "Future of scrum: Parallel pipelining of sprints in complex projects", in *Proceedings of Agile Conference 2005 (AGILE 2005),* Denver, Colorado: IEEE, pp. 90-99.

Sutherland, J., Schoonheim, G. & Mauritz, R. 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, J., Viktorov, A., Blount, J. & Puntikov, N. 2007, "Distributed scrum: agile project management with outsourced development teams", in *Proceedings of the 40th Hawaii International Conference on System Sciences (HICSS-40),* Big Island, HI: IEEE, pp. 274a-274a.

Svahnberg, M., Gorschek, T., Feldt, R., Torkar, R., Saleem, S.B. & Shafique, M.U. 2010, "A systematic review on strategic release planning models", *Information and Software Technology,* vol. 52, no. 3, pp. 237-248.

Syed-Abdullah, S., Holcombe, M. & Gheorge, M. 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, H. & Nonaka, I. 1986, "The New New Product Development Game", *Harvard business review,* vol. 64, no. 1, pp. 137-146.

Tengshe, A. & Noble, S. 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.

Vähäniitty, J. 2004, "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, J., Rautiainen, K. & Lassenius, C. 2010, "Small software organizations need explicit project portfolio management", *IBM Journal of Research and Development,* vol. 54, no. 2, pp. 1:1-1:12.

Vähäniitty, J. & Rautiainen, K. 2008, "Towards a conceptual framework and tool support for linking long-term product and business planning with agile software development", in *Proceedings of the 1st international workshop on Software development governance (SDG '08),* Leipzig, Germany: ACM, pp. 25-28.

Vähäniitty, J. & Rautiainen, K. 2005, "Towards an Approach for Development Portfolio Management in Small Product-Oriented Software Companies ", in *Proceedings of the 38th Hawaii International Conference on System Sciences (HICSS-38),* Big Island, HI: IEEE, pp. 314-314.

Valkenhoef, G.,van, Tervonen, T., Brock, B., de & Postmus, D. 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.

VersionOne Inc. 2009, *4th Annual "State of Agile Development" Survey*, http://pm.versionone.com/StateOfAgileSurvey.html.

Vlaanderen, K., Jansen, S., Brinkkemper, S. & Jaspers, E. 2011, "The Agile Requirements Refinery: Applying SCRUM Principles to Software Product Management", *Information and Software Technology,* vol. 53, no. 1, pp. 58-70.

Vlaanderen, K., Jansen, S., Brinkkemper, S. & Jaspers, E. 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.

Wallin, C., Ekdahl, F. & Larsson, S. 2002, "Integrating business and software development models", *IEEE Software,* vol. 19, no. 6, pp. 28-33.

Wangenheim, C.G.,von, Weber, S., Hauck, J.C.R. & Trentin, G. 2006, "Experiences on establishing software processes in small companies", *Information and Software Technology,* vol. 48, no. 9, pp. 890-900.

Weerd, I.,van de, Brinkkemper, S., Nieuwenhuis, R., Versendaal, J. & Biljsma, L. 2006, "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.

Wheelwright, S.C. & Clark, K.B. 1995, *Leading product development: the senior manager's guide to creating and shaping the enterprise,* New York, NY: The Free Press.

Wheelwright, S.C. & Clark, K.B. 1992, "Creating Project Plans to Focus Product Development", *Harvard business review,* vol. 70, no. 2, pp. 70-82.

Zika-Viktorsson, A., Sundström, P. & Engwall, M. 2006, "Project overload: An exploratory study of work and management in multi-project settings", *International Journal of Project Management,* vol. 24, no. 5, pp. 385-394.

The editors, (from left) Ville Heikkilä, Jarno Vähäniitty and Kristian Rautiainen, relaxing in Volendam after a hard day's work.

We hope you enjoyed reading the book as much as we enjoyed writing it!

Aalto University's Software Process Research Group (SPRG) offers research collaboration, evidence-based training and consultancy for a sober price.

Contact us to learn more at sprg@soberit.hut.fi.

ISBN 978-952-60-3498-0