Hans-Bernd Kittlaus · Samuel A. Fricker

Software Product Management

The ISPMA-Compliant Study Guide and Handbook

Springer

Software Product Management

What first readers of "Software Product Management: The ISPMA-Compliant Study Guide and Handbook" say:

"Software-intensive products are at the heart of many businesses, so product management is a paramount business activity. But how can businesses be perfect at it? This book is the answer. It is your vade mecum for all product management topics and aspects."

Dr. Karl Michael Popp, Chief Product Expert and Director Corporate Development, SAP SE, Walldorf, Germany.

"A book that goes beyond platitudes and offers concrete methods and frameworks to product managers working with software intensive product development. The authors have a sound footing in both practice, but also state-of-the-art research, and manage to combine the two."

Prof. Dr. Tony Gorschek, Blekinge Institute of Technology, Karlskrona, Sweden.

"By reading and applying the lessons from the new book "Software Product Management: The ISPMA-Compliant Study Guide and Handbook" (by Hans-Bernd Kittlaus and Samuel A. Fricker), you will gain strategic advice, practical techniques, and great insights into how to accelerate software product management (SPM) success. These practices and methods will be useful to executives and practitioners in this demanding area—and those who aspire to a level of software excellence—in both their current jobs and future careers, as they work to help organizations deliver superior value and effective software solutions to a world of increasingly demanding customers."

Michael Eckhardt, Managing Director, CHASM Institute, Palo Alto, California, USA.

"Call her/him a linchpin, a rudder or even a conscience keeper, the role of Software Product Manager (SPM) is extremely critical for the viability as well as sustainability of any software product business. This book is a "must read" as well as "must have", not just for every SPM but for all the key stakeholders and decision makers connected with a software product business. And for all the business leaders in the software services industry aspiring to extend their success into software product business, here's your definitive reference."

Haragopal Mangipudi, CEO, finUNO, Bangalore, India (fintech startup) (formerly Infosys SVP and Global Head of Finacle).

"Software has been turning into the dominating value driver in most traditional industries like automotive and banking. I recommend this book not only to software professionals, but also to managers in these other industries. It provides comprehensive structural and operational help how to set up and run product management of software-intensive products. Particularly fascinating are the authors' highly topical

ideas to extend the discipline of SPM into the management of industrial ecosystems to tackle the increasing complexity with an integrated consistent approach."

Wilhelm Gans, CTO, DSV Group (German Savings Banks Organization), Stuttgart, Germany (banking industry).

"This book provides a comprehensive and enlightening knowledge foundation for both practitioners and researchers in the increasingly important domain of software product management."

Prof. Dr. Björn Regnell, Lund University, Sweden.

"This compendium based on industry best practices provides a toolbox for software product managers and executives to ensure sustainable success of software products along their life cycles. The authors have vividly described the multi-faceted role of a software product manager—the mini CEO—and his embedding into the corporate organization. Special emphasis is put on areas relevant for SPM such as pricing models, legal aspects, ecosystem management and orchestration that are not covered too well in the available literature. A must-read for everyone interested in the software business aspects in all industries."

Michael Conrad, Director Portfolio Management, AVL List GmbH, Graz, Austria (automotive industry).

Praise for "Software Product Management and Pricing," Hans-Bernd Kittlaus' previous book which continues to be the only extensive publication on software pricing:

"These two seasoned practitioners have masterfully distilled the essence of the software business and the art and craft of the increasingly important and challenging field of software product management. Worthwhile to any who want an appreciation of the evolving world of product management, seasoned veteran and new entrant alike."

Richard Campione, Senior Vice President, Business Suite Solution Management and CRM On Demand, SAP, Germany/USA.

"Mr. Kittlaus and Mr. Clough have used their considerable knowledge and experience to succinctly lay out the value chain that is essential to the development of a financially healthy software company. If you want to understand how to turn software technology into a long-term profitable company this is the book to read."

Paul Kaplan, Vice President, Worldwide Enterprise Software Sales, Software Group, IBM, USA.

"This book on Software Product Management and Pricing is the first book that treats the business of software in a systematical way. Although software products were already shipped in the seventies of the last century, there are hardly any books providing an overview of all issues a company faces when playing a role in this industry. Product management and pricing are key processes, and this book informs the reader of the essentials. It is a must-read for anyone involved in software products, be it in business or in research."

Prof. Dr. Sjaak Brinkkemper, Information and Computing Sciences, Utrecht University, Netherlands.

Hans-Bernd Kittlaus • Samuel A. Fricker

Software Product Management

The ISPMA-Compliant Study Guide and Handbook

Springer

Hans-Bernd Kittlaus
InnoTivum
Rheinbreitbach, Germany

Samuel A. Fricker
University of Applied Sciences
Northwestern Switzerland
Windisch, Switzerland

Blekinge Institute of Technology
Karlskrona, Sweden

ISBN 978-3-642-55139-0 ISBN 978-3-642-55140-6 (eBook)
DOI 10.1007/978-3-642-55140-6

Library of Congress Control Number: 2017938640

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer-Verlag GmbH Germany
The registered company address is: Heidelberger Platz 3, 14197 Berlin, Germany

Preface

The authors are very grateful to all our contributors without whom we would not have been able to write a book as comprehensive as this. Here is some information on authors and contributors:

Authors

Hans-Bernd Kittlaus is the owner and managing director of InnoTivum Consulting (www.innotivum.com) and works as consultant, interim manager, and trainer for software organizations, in particular in the areas of software product management and organizational aspects of software organizations. Before, he was Director of SIZ GmbH, Bonn, Germany (German Savings Banks Organization) and Head of Software Product Management and Development units of IBM. He has published numerous articles and books, e.g., "Software Product Management and Pricing—Key Success Factors for All Software Organizations," Springer, 2009. He is Diplom-Informatiker, ISPMA Certified Software Product Manager, Certified Scrum Product Owner (CSPO), Certified PRINCE2 Practitioner, and member of ACM and GI. Hans-Bernd lives near Bonn, Germany. He is a founding board member and current chairman of ISPMA. Contact: www.innotivum.com, email: hbk@innotivum.de, Blog: www.innotivum.com/publications/spm-blog/

Samuel A. Fricker is a Professor of Requirements Engineering at the University of Applied Sciences and Arts Northwestern Switzerland (FHNW) and an Assistant Professor of Software Engineering at Blekinge Institute of Technology (BTH). He received his Ph.D. degree in 2009 from the University of Zurich. Samuel spent his career in both industry and academia. Important industry stays were with Ericsson, ABB, and Zuehlke. He researches socio-technological alignment mechanisms for software and networked systems. Samuel leads FHNW in the EU Horizon 2020 projects Wise-IoT, Bonseyes, and SMESEC and is a researcher in the Horizon 2020 project SUPERSEDE. He led BTH in the EU FP7 project FI-STAR. Samuel lives in the greater Zurich area in Switzerland. He is a founding member, former chairman, and current board member of ISPMA. Contact: bit.ly/sfr_fhnw, email: samuel.a.fricker@gmail.com, Twitter: @samuelfricker

Contributors

Peter Clough was Senior Vice President and management consultant with InnoTivum Consulting specializing in software pricing. Previously, he was Enterprise Software Sales Executive and Manager of Software Offerings in IBM's Software Group and influenced, participated in, or managed every major IBM software terms' development between 1983 and 2008, after holding previous positions in IBM's hardware and software marketing and sales. Peter lives in New York, USA. He served as reviewer of major parts of this book. Contact email: pnc@innotivum.com

Gerald Heller is principal consultant of Software.Process.Management with more than 30 years of experience in large-scale, global distributed software development. During his professional career, he covered almost every aspect of the product development life cycle. He holds a diploma in computer science and is a member of GI and ASQF. Gerald lives near Stuttgart, Germany. He is a founding member and current board member of ISPMA. Gerald served as reviewer of parts of this book. Contact email: gerald.heller@swpm.de

Barbara Hoisl is an independent consultant and trainer, supporting software vendors and Internet companies. Her work is based on more than 20 years of industry experience, including 14 years with the software product business of HP. She holds a master's degree in Computer Science with a minor in Business Administration from Technical University of Kaiserslautern. Barbara lives in Stuttgart, Germany. Barbara is a fellow member of ISPMA. She contributed text to Chap. 5 and served as reviewer of parts of this book. Contact: www.barbarahoisl.com, email: info@barbarahoisl.com

Peter Lick is process and skill manager for product management at AVL, a competent partner to the automotive industry. His professional experience comprises portfolio and innovation management, marketing and software product management for process, and testing software. Peter holds an academic degree in electrical engineering and automation from the Technical University of Graz and additional degrees in coaching, marketing, and management. Peter is a board member of ISPMA. He served as reviewer of parts of this book. Contact: www.avl.com, email: peter.lick@avl.com

Garm Lucassen is a Ph.D. student at Utrecht University in the Netherlands. His research focuses on the complex relationship between software product managers and software architects. His current research efforts focus on why user stories are an effective approach to expressing requirements and how to help practitioners create higher quality user stories. Together with Prof. Dr. Sjaak Brinkkemper, he coordinates and teaches the SPM foundation level course in Utrecht. Garm is a board member of ISPMA. He served as reviewer of parts of this book. Contact: www.garmlucassen.nl, email: G.Lucassen@uu.nl

Andrey Maglyas is a postdoctoral researcher having finished his D.Sc. at Lappeenranta University of Technology, Finland. His research interests include empirical investigation of software product management practices, methods, and tools. In addition, he is an active member of product management community in Finland and Russia and cofounder of ProductCamp Helsinki. Andrey lives in St. Petersburg, Russia. He is a board member of ISPMA. He has contributed text to Chap. 7 and served as reviewer of parts of this book. Contact email: maglyas@gmail.com.

We would like to thank ISPMA e.V. as an organization for allowing us to make use of ISPMA's published material. In addition to the colleagues listed above, we also thank all the other ISPMA members who contributed to ISPMA's syllabi: Jonas Als, Magnus Billgren, Erik Bjernulf, Prof. Dr. Sjaak Brinkkemper, Prof. Dr. Christof Ebert, Prof. Dr. Tony Gorschek, Rainer Grau, Prof. Dr. Georg Herzwurm, Dr. Marc Hilber, Robert Huber, Dr. Slinger Jansen, Dr. Mahvish Khurum, Daniel Lucas-Hirtz, Lars Olsson, Dr. Katharina Peine, Dr. Karl Michael Popp, Greg Prickril, Prof. Dr. Guenther Ruhe, Ramesh Sundararaman, Dr. Kevin Vlaanderen, Dr. Inge van de Weerd, and Dr. Krzysztof Wnuk.

Though we acknowledge the contributions of all these colleagues, the contents of this book including any mistakes, omissions, and judgments are the sole responsibility of the authors.

We would also like to recognize Peter Clough, Christoph Rau (†), and Juergen Schulz, Hans-Bernd's coauthors of his previous books on software product management [KittClou09, KiRaSch04] for all the work that they put into them and that helped with this publication. We thank Hermann Engesser and Dorothea Glaunsinger from Springer-Verlag for the good cooperation. We thank Sjaak Brinkkemper, Utrecht University, Netherlands; Inge van de Weerd, Free University of Amsterdam, Netherlands; and Gartner, Inc. for giving us some graphical images for publication. Thanks also to Strategic Pricing Group, Thomas Nagle, and John Hogan for allowing us to reproduce their Strategic Pricing Pyramid. Last but not least, special thanks go to our better halves for their moral support.

Rheinbreitbach, Germany — Hans-Bernd Kittlaus
Oberuzwil, Switzerland — Samuel A. Fricker
December 2016

Contents

1 Introduction 1
- 1.1 About this Book 3
- 1.2 Conventions 4

2 Management of Software as a Business 7
- 2.1 A Little History 7
- 2.2 Product Management for Software: Terms and Characteristics 10
- 2.3 Software as a Business 16
- 2.4 Business Models 21
- 2.5 The Software Product Management Framework 33
- 2.6 The Role and Organization of SPM 40

3 Product Strategy 49
- 3.1 Product Vision 51
- 3.2 Product Name 56
- 3.3 Customers 58
- 3.4 Market 59
- 3.5 Product Definition 67
- 3.6 Positioning 71
- 3.7 Service Strategy 77
- 3.8 Sourcing 79
- 3.9 Business View 82
- 3.10 Pricing 91
- 3.11 Ecosystem Management 96
- 3.12 Legal Aspects 101
- 3.13 Performance and Risk Management 112
- 3.14 Product Strategy Processes and Documentation 116

4 Product Planning 119
- 4.1 Roadmapping 120
- 4.2 Product Requirements Engineering 130
- 4.3 Release Planning 157
- 4.4 Product Life Cycle Management 173
- 4.5 Process Measurement and Improvement 181

5 Strategic Management 189
5.1 Corporate Strategy 190
5.2 Portfolio Management 196
5.3 Innovation Management 202
5.4 Resource Management 207
5.5 Market Analysis 208
5.6 Product Analysis 215
5.7 Corporate Strategy Processes 218

6 Orchestration of the Organization's Functional Areas 219
6.1 Role and Processes 221
6.2 Development and UX Design 223
6.3 Marketing 230
6.4 Sales and Distribution 237
6.5 Service and Support 243
6.6 Orchestration Skills 249

7 SPM Today and Tomorrow 255
7.1 The Future of SPM 255
7.2 The State of Practice 258
7.3 SPM in Different Business Scenarios 261
7.4 ISPMA 265

Glossary 267

Bibliography 275

Index 289

Introduction 1

Why do we write another book on software product management (SPM)? And what is so special about product management for software compared to other products?

Product management is a discipline that has been utilized by many industries for decades, above all the consumer goods industry. The invention of product management as an explicit management concept is attributed to Procter and Gamble. In 1931, the company assigned one product manager to each of two competing soap products (see [Gorche11]). Since then, this basic idea has become widespread. In fact, it makes sense for any company to manage explicitly the products that generate its revenue and that, as assets, represent the company's sustainable value. But what does product management mean? Unfortunately, a general answer can only be given for parts of this question. The activities of the product manager depend largely on the type of product involved, the culture, history, and organization of the company, and the target and reward system. Product management means planning and coordinating all relevant areas of a product inside and outside the company with the aim of sustainably optimizing product success.

The focus of this book are the tasks of a software product manager, and how they can be performed well. We follow a best-practice approach and make use of any techniques that help product managers, from whichever methodology they come, be it lean, agile, kanban etc. In our experience, methodologies come and go over time, the tasks remain. Therefore, we do not use temporarily marketable terms like "Agile Product Management" or "Lean Product Management". The prime objective of a product manager is not to follow any particular fashionable methodology, but to take care of his tasks in the best possible way, and thereby make his product successful over its life cycle.

Software has become highly pervasive. There is hardly any industry that is not increasingly dependent on software, be it as part of their products or as the backbone of their business operations. A good example of that phenomenon is the automobile industry. The software contents of cars have increased significantly over the last 10 years. In more and more industries software is turning into the number one value driver.

H.-B. Kittlaus, S.A. Fricker, *Software Product Management*,
DOI 10.1007/978-3-642-55140-6_1

Another example is corporate IT organizations. For them, it has become apparent that they need to manage their software applications explicitly as crucial assets with a life cycle perspective, i.e. as software products. The realization that companies from non-IT industries are suddenly becoming "standard software suppliers" by providing apps to their customers may also have contributed to this development.

Software has become important also in our private sphere. Many use software to manage their private lives and to communicate with others. Software has also started to be critical for addressing the big challenges of our societies. For example in the healthcare area, internet-based apps empower patients and their caregivers to make decisions for themselves. This empowerment allows expert healthcare staff to serve more patients for less cost per patient.

Software product management—as we define it—does apply to all these organizations:

- Vendors of software products and software-intensive technical services, be it licensed products, or Platform-as-a-Service (PaaS) or Software-as-a-Service (SaaS) including apps running on all kinds of smart devices (software industry),
- Vendors of software-intensive products (all industries), e.g. cars or smartphones,
- Vendors of professional human services (all industries) in which software is used to increase productivity, and
- Corporate IT organizations (all industries).

In several decades of experience in the software industry, the authors have come to realize that the knowledge and experience acquired in other business areas and with other types of products are only partially transferable to software. We believe that software is the most complex product of human invention that we know of (see Chap. 2). That complexity implies that the management of software products is special and puts unique demands on the persons responsible for this task (see also Sect. 2.6).

A software product manager's job is special because software is special, i.e. the characteristics of software are different from most other products and have a strong influence on what a software product manager does. Most conspicuous are

- High frequency of change over the life cycle of the software with the resulting great importance of requirements management.
- High complexity.
- Ability to interact with customers through the product.
- Flexibility of re-configuring the product and adapting it to new purposes and usage contexts.
- Culture of searching for lightweight, agile approaches to building and evolving software.
- No or little need for physical manufacturing and distribution.
- Increasing returns through network effects.
- Special financial picture due to low marginal cost.

When Hans-Bernd Kittlaus published his first book on Software Product Management [KiRaSch04] and his second book on "Software Product Management and Pricing" [KittClou09], there were not too many publications on this subject available. Since then, the situation has changed significantly. The number of SPM-related publications has increased, there is more focused research in academia, and there is an increasing number of commercial SPM training offerings available.

These changes are to some degree due to the establishment of International Software Product Management Association (ISPMA, www.ispma.org). First convened in 2009, ISPMA is a non-profit organization whose fellow members are SPM experts from the industry and academia. Other types of membership are open to all interested companies and individuals. ISPMA's goal is to foster software product management excellence across industries (for more information see Sect. 7.4). ISPMA's fellows have developed a curriculum and a Certifiable Body of Knowledge (SPMBOK) that have become the basis for a high number of commercial training offerings and university courses. Samuel Fricker and Hans-Bernd Kittlaus are co-founders of ISPMA. Hans-Bernd is ISPMA's current chairman, Samuel is ISPMA's former chairman and current board member.

1.1 About this Book

This book is intended to provide an integrated view of software product management for all the organizations and scenarios listed above. We consider the similarities between the scenarios that are induced by the specifics of software as more significant than differences in organizational views. Nevertheless, there are some differences that require different priorities regarding the individual tasks (see Sect. 7.3).

The target group of this book is everyone involved or interested in software product management, be it on the academic side or in organizations in the scenarios listed above.

With this book, the authors intend to provide a state-of-the-art update of the SPM part of [KittClou09] that is compliant with ISPMA's Body of Knowledge[1] (as of December 2016):

- ISPMA SPM Foundation Level Syllabus V.1.3
- ISPMA SPM Excellence Level Syllabus Product Strategy V.1.1
- ISPMA SPM Excellence Level Syllabus Product Planning V.1.1
- ISPMA SPM Excellence Level Syllabus Strategic Management V.1.2
- ISPMA SPM Excellence Level Syllabus Orchestration V.1.0.

[1] www.ispma.org

As a result, the reader can use the book for supporting the preparation of an examination for ISPMA certification. However, the primary source for preparation is always the current release of the corresponding syllabus and the corresponding training.

We introduce the ISPMA software product management (SPM) framework in Chap. 2. The remaining chapters will follow that structure. The subject of software pricing will be addressed in this book (see Sect. 3.10), but not as extensively as in [KittClou09].

Due to the wealth of publications related to SPM over the last couple of years, the authors do not claim to provide a comprehensive bibliography. We have rather restricted ourselves to publications most often cited and publications we find particularly useful. By reading this book, the reader will know the views of the most influential results of research aimed at SPM topics.

Clear boundaries are needed to be able to address the breadth of the topic of software product management. Despite important interfaces between development and software product management, neither the topics of software development management nor project management will be discussed here. We have also omitted operations and consulting issues: the book does not cover bookkeeping, administration, and payment for the software licenses acquired and used in a company. Also, it does not cover software product line management, i.e. the management of development variants based on the same product platform.

The book is structured as follows. Chapter 2 looks at software as a business and how we suggest to manage it in terms of business models and organizational aspects. We define relevant terms and introduce the ISPMA SPM Framework. Chapter 3 describes the tasks and activities related to product strategy. Chapter 4 focuses on product planning. Chapter 5 covers strategic management. Chapter 6 describes the orchestration of the organization's functional areas. Chapter 7 rounds things up by looking at SPM's state-of-practice, the application of SPM in different business scenarios, as well as the future of SPM.

At the end of the book, we provide a glossary, i.e. a list of definitions of all relevant terms used. It is aligned with ISPMA's glossary as of December 2016.

1.2 Conventions

Before we begin our in-depth discussion of SPM, we need to clarify several conventions used throughout this book. Terms such as "manager" or "director" are meant to be gender-neutral, that is they refer equally to female and male persons. While women increasingly participate in IT management, we have chosen a convention of referring to such positions using the male pronoun. This is not intended to be in any way discriminatory and was chosen simply for the purposes of easier readability. We use terms like development or marketing with small characters when we mean the activity, with capital characters when we mean the organizational unit.

We use the term "software vendor" when we mean companies whose primary business is the development and provision of software for commercial purposes for a relatively large number of consumer and business customers. Examples are Microsoft, SAP, Oracle, IBM, Google, Samsung and Huawei. We use the term "corporate IT organization" for organizational units that are part of companies in all industries and whose primary mission is to provide software or IT support for the parent company or corporation, which we call "corporate customer". In that sense, software vendors are corporate customers of their internal IT.

The term "service" has many different meanings (see Webster's Dictionary). The following three meanings are relevant to this book:

- Useful labor that does not produce a tangible commodity (as in "professional services").
- A provision for maintenance and repair (as in "software maintenance service").
- The technical provision of a function through a software component that can be accessed by another software component. Such access often occurs over a network and executed on a remote server (as in "web services", "Software-as-a-Service", or "Service-Oriented Architecture").

Whenever we use the term "service" in this book, we try to make it clear which meaning is intended.

When we use the term "controlling" we mean "performing the functions of a (business or financial) controller".

And now, we can jump right into the secrets of software product management.

Management of Software as a Business

2

Product management has become an established discipline in many industries since Procter and Gamble introduced it in 1931. During the last decades, most software product companies—such as Microsoft, IBM, and Google—have implemented Software Product Management (SPM). So did a few corporate information technology organizations in essentially all industries, as well as some companies that produce software embedded in software-intensive products and services. The role of software product manager has emerged during this time as being of strategic value since it is crucial to the economic success of a product.

This chapter puts software product management into the context of software as a business. It outlines the historical context, defines relevant terms and characteristics, looks at software from a business perspective, introduces ISPMA's SPM Framework, and discusses the role of SPM and its positioning in a software organization. The business context, definitions, and scenarios will help the reader to understand how to implement the many practices that a software product manager has available to him to ensure the success of his product.

2.1 A Little History

To understand the business aspects of software, a short detour into the history of this young industry is helpful before we analyze the current business drivers in more detail.

The term "software" was first coined in a 1958 article by Tukey from AT&T's Bell Labs [Tukey58]. The term has become an integral element of the English language and has been taken over by many other languages. At the beginning of the computer era, the computer was considered as a mere machine. Just as we tell an automobile motor to run faster by pressing the gas pedal, the computer was told what to do one instruction at a time. Manufacturers sold the computer as a physical machine and added the operating system and rudimentary software at no additional cost. Not before the early 1960s was software considered independent and separate,

H.-B. Kittlaus, S.A. Fricker, *Software Product Management*,
DOI 10.1007/978-3-642-55140-6_2

and at the same time entrepreneurs started to see an opportunity for a software products business ([Cusuma04], p. 90).

Several technical developments triggered this decoupling of software programs from the machine. In the beginning, the instruction sets were computer-specific, i.e. each processor had its individual assembler language. That is why programs were tied to the respective processor and could not be ported to any other processor. From the late 1950s on, the first higher-level programming languages like FORTRAN and ALGOL were created that enabled programming on a more abstract logical level. Compilers that transformed high-level source code into different assembler languages were offered for these languages. High-level languages enabled the programmer to develop programs for different processors. Porting a program from one processor to another came into reach, even though Java's slogan "Write Once, Run Everywhere" was still considered a vision when it was introduced in the mid-1990s. IBM released the/360 processor series in 1964 that contained processors at a broad spectrum of performance points that could be programmed with the same assembler language. This development allowed the decoupling of software from the machine, the understanding of software as something independent, separate. Pressed by the US Department of Justice and facing forthcoming antitrust lawsuits, IBM announced on June 23, 1969, that it would unbundle hardware and software in the future. This announcement can be seen as the birth date of the software industry as we know it today. Within a few decades, it has developed into an enterprise software market with a volume that Gartner estimates at 314 billion US$ in 2015, with an annual growth rate (CAGR) in the range of 6.8% [Gartner16].

In spite of the overwhelming success of software as an industry, there is an immense risk in the software business. Compared to more traditional industries, the software industry is characterized by high speed, a rapid succession of technological changes and a lot of irrational hypes. In some aspects, it is still quite immature. When the term "software engineering" was coined at a NATO conference in Germany in 1968, the intention was to bring the reliability and scientific approach of other engineering disciplines to software development. However, even today the failure rate of software development projects is still considerably higher than in those other engineering disciplines. Nevertheless, software has pervaded most other industries to an amazing—and sometimes frightening—degree.

Over the last 50 years, the world of business both in the developed and in the emerging economies has gone through change at a pace and extent of impact mankind has never experienced before. Over the last 20 years, the pace has increased even further. Most of this change is enabled if not driven by information technology. The fundamental drivers behind that are Moore's Law and the Internet.

2.1.1 Moore's Law

In 1965, Gordon Moore observed that the number of transistors that could be placed on an integrated circuit was doubling every 2 years. Since the mid-70s, it has been doubling every 1.5–2 years, currently slowing down a bit. Recently this development could not

be translated into speed increases of single CPUs anymore, which is the reason for why we see multi-core architectures that continue to provide significant performance improvements when compared at the same price level. For storage, the price per storage unit has been going down by a factor of 2 per year. These exponential improvements have led to today's situation that people have much more processing power and storage capacity in their smartphones than a mainframe computer in a water-cooled computing center had 40 years ago. Processors and storage, as well as software, are embedded in all kinds of products like cars, cameras, or mobile phones. Some industries have been totally transformed by this shift from analog to digital. Though Moore's Law does not hold anymore on the chip level, there continues to be good reason to expect further exponential growth in computing for the foreseeable future [DenLew17].

2.1.2 The Internet

Originally being the military communications network ARPAnet in the US in the late 1960s, the internet has turned into the world's premier communications infrastructure. Hundreds of millions of people use it all over the world. It enables them to communicate via email, instant messaging and other paths. Since the introduction of HTML and the web browser in the mid-90s, people have been able to access information with unprecedented ease and speed. The internet has given rise to totally new applications and business models. Examples are search engines like Google Search, online auctions like eBay, social networks like Facebook, internet banking, brokerage, and retail platforms like Amazon.

All industries make extensive use of IT today. IT may be an integral part of products, a major factor of development and manufacturing systems, or the backbone of business systems. Banking, financial services and music are just some examples of industries that have already been fundamentally influenced and radically changed due to IT and in particular software:

- The worldwide financial markets are totally dependent on IT, in terms of both size and speed. Prices and returns on many innovative financial products can only be calculated with the help of IT. This dependency on IT led to the fact that some players in the industry consider themselves more as IT companies than banks.
- When the music industry switched from LP to CD, i.e. from analog to digital, in the early 80s, they did not foresee the consequences. In the digital domain, a copy is identical to the original whereas in the analog domain it is not. With the proliferation of PCs and the internet, making digital copies and distributing them has become as easy as a mouse click. The music industry has been helplessly watching its decline. The pinnacle of humiliation was the success of Apple, an IT company, with iTunes, the music download platform that demonstrated how to be successful with music in the digital domain.

Before we look at the current drivers of software as a business, we need to define our vocabulary.

2.2 Product Management for Software: Terms and Characteristics

Software is an intangible economic good, with no physical form. Only the functionality is perceptible, e.g. via a user interface, or visible in the results of transactions controlled via software, e.g. as account movements. As with many highly technical products, many people do not understand how software products work. Software is therefore in the truest sense of the word "intangible." Software thus contrasts greatly with other business investments or acquisitions of consumer goods. In particular, the customer does not acquire the product when buying software but specific, precisely defined rights of use as specified in a license or a Software-as-a-Service (SaaS) contract. However, investments in software today represent a larger proportion of spending for IT infrastructure than investments in hardware. Software contracts with large companies often amount to multiple millions of dollars.

Software belongs not to the three classic economic factors of capital, land, and labor, but in the new fourth category of "knowledge." Software is the manifestation of human know-how in bits and bytes. This form possesses the invaluable advantage that software can be easily copied and quickly circulated over any distance—which can also be a disadvantage as the example of the music industry above demonstrated. The right software, used appropriately, can represent a more important strategic competitive advantage in today's economic life than all the other factors. Software can be crucial for competitiveness in production processes, functionality, the availability of service products, and thus for a company's success or failure on the market. But what is a software product and what is it not—or not yet? What role does the price play? How should we classify services offered on the basis of software?

We find it reasonable not to limit the term "software product" to the world of software vendors, but also to use it in the world of corporate IT organizations. Is, for example, an online account with a direct bank, a mobile phone in combination with a special mobile phone tariff or the membership to a chat community a software product? In this chapter, we attempt to define the term "software product" and to discuss certain features of software and their relevance for products and product management.

Marketing defines the term "product" as follows: "a product is anything that can be offered to a market for attention, acquisition, or consumption that might satisfy a want or need." (see [KotArm15, p. 256]).

While products typically address bigger markets, if not "mass markets", we want to include internal customer-supplier relationships as well. Therefore we put the relationship between two parties in the foreground.

Product = A combination of goods and services, which a supplier or development organization combines in support of its commercial interests to transfer defined rights to a customer.

Combination of material or intangible goods and services, which one party (called vendor) combines and evolves in support of their commercial interests,

with the intention to transfer defined rights to one or more second parties (called customers).

Software product = Product whose primary component is software.

The phrase "in support of their commercial interests" should make clear that it refers to business but does not necessarily lead to payment. There is also a commercial interest behind Open Source. Even a product free of charge (e.g. Adobe's Acrobat Reader) has a commercial goal—to increase market penetration of another product from the same software vendor that is subject to a fee. The phrase "defined rights" expresses that there is some room for variation here, e.g. right of use (possibly with restrictions), property right, and right of resale. Details are typically defined in the software vendor's licensing terms or an individual contract between the parties concerned.

This product definition should also be construed as meaning that an item may already be a product when it has not yet been bought. It does not become a product through the buying process, but through the intention to make it available as a useful entity to third parties, either inside or outside one's corporation, for monetary consideration or not.

In establishing the boundaries of what a software product is and is not, we have knowingly chosen a "flexible" phrase: the word "primary" should make it clear that there is room for discretion.

Still, a mobile phone is not a software product according to our definition (rather a telecommunication product), even if software is an important part and may have absorbed a large proportion of the development costs. In this case, we would be talking about embedded software. Here, embedded software "serves" most of the functionality and is, therefore, an underlying part of the whole product. The embedded software cannot be bought separately. We define:

Embedded software = Software parts of software-intensive systems that are not marketed and priced as separate entities.

Embedded software does not manage and operate only a computer or a processor, but rather is embedded in a technical system that consists of other components. All components together allow the whole thing to become a product. Embedded software can be the software for programming and managing a machine, diagnosis software for finding errors in an automobile, or software for servicing a dialysis machine in medicine. These programs serve highly specialized interfaces, which are typically very closely integrated with hardware.

The requirements and the product management for embedded software are driven by the functionality of the complete system that we call a "software-intensive system". Software-intensive systems can be products from many industries like cars, airplanes, and smartphones. Software-intensive services, often delivered as cloud or internet services, can also be products from any industries like financial, insurance, gaming, social software, or personal services based on software support. For the ease of reading this book, we will use the term "software products" instead of explicitly referring to software-intensive products as well.

Another important term in this context is "OEM product". OEM stands for Original Equipment Manufacturer. We define:

OEM software product = software product of software vendor A that is used by company B as a component under the covers of one of B's products.

The term "OEM" was originally coined in the hardware business and later transferred into the software world. It means that one manufacturer sells one of his products to another manufacturer who uses it as a component in one of his products without showing its origin openly. We differentiate "OEM" companies from integrator companies, the former offering products to customers, the latter a system development service to a specific customer.

Notice that B's product can be, but does not have to be a software product. A vendor is usually willing to sell his products as OEM products at a significantly reduced price to increase his volume.

Let's look at some more examples. A games console is also not a software product (rather a games product). The same arguments are valid here as for the mobile phone. A game for this console—purchased separately, separately packaged with its price and own terms of licensing—is obviously a software product, however.

An online bank account is not a software product, according to this definition. Rather, the account is a banking product implemented with the help of software products. The bank's customer receives online access to his account from his bank, which allows him to carry out bank transactions (bank balance enquiries, transfers, establishment of standing orders etc.) at home or wherever he happens to be. The software is only useful and usable in connection with the account. So the account is the primary component.

A search offering like Google qualifies as a software product even though that may be contrary to some people's intuitive understanding. It does fulfill all the criteria of our definition: there is a commercial interest, the customer gets the right to use it, and its primary component is software. This approach builds on the "Software as a Service" model (SaaS).

The term "solution" is quite popular from a marketing perspective since it implies that a customer's problems are addressed and solved. It is used by both product vendors and professional service organizations.

Solution =

(a) A product that is a combination of other products, human services, and possibly some glue code and customization.
(b) A combination of products and customer-specific code that is developed and implemented for a specific customer.

An example for (a) are SAP's solutions for different industries or lines of business.

2.2.1 Product Platform, Family, and Line

For technology companies, it can make sense to differentiate between product platform, product family, and individual product. McGrath defines [McGrat01]: "A product platform is not a product. It is a collection of the common elements, especially the underlying defining technology, implemented across a range of products. A product platform is primarily a definition for planning, decision making, and strategic thinking. The choice of a defining technology in platform strategy is perhaps the most critical strategic decision that a high-technology company makes." So we define:

Product Platform = the technical foundation on which several software products are based.

An example of a product platform is the SAP core system that serves as the basis for all SAP components. With software, there can be platforms that are not under the control of the software vendor, but which nevertheless may have a key influence on the success of his product. For example, for any smartphone software the question of the operating system platforms needs to be answered. I.e. a vendor must define whether the software shall run on Apple's iOS, Google's Android, Microsoft's Windows, or other platforms such as Firefox OS.

A product platform is not necessarily an independent product, but rather a combination of technological elements used in various products. Such a platform often constitutes a valuable asset and serves as a market differentiation factor. Therefore, the platform requires cautious management, since errors will immediately have serious consequences for all products based on the platform and thus for the company as a whole. An interesting example is Amazon, which not only developed a highly successful e-commerce platform, but also a second platform with Amazon Web Services (AWS).

A platform that brings real competitive advantage may serve as a base for a whole family of products. The establishment of a product family in the market, however, is motivated by pure marketing reasons. An example is IBM's DB2 family. In the 1990s, IBM combined all relational database products on the various system platforms to the DB2 product family and gave all products this name (which had previously only been used for the host product). However, this was not based on a joint code base of the individual products, even though this would have been preferable from a development and customer perspective. In any case, customers associated the same family name with a large degree of similarity. We define:

Product Family = A group of software products that are marketed as belonging together under a common family name.

A software vendor groups various products together under a "family name". That family name allows marketing the products more efficiently than with single-product marketing approach. This product family approach suggests that the products belong together, implying that they are either technologically similar or that together they provide a solution for specific problems. The technological similarity can be a common product platform, e.g. SAP products, or a common basic technology, e.g. IBM's DB2 family. Microsoft Office is an example of a product family comprising a group of products that address specific problems, in this case, office tasks, even if the components of Office have not always had a seamless relationship.

The above examples illustrate that the terms "product platform" and "product family" sometimes coincide. Products based on a common product platform can be—but do not have to be—marketed as a product family. Conversely, products marketed as a family can have—but do not have to have—a common product platform. Whether or not it is advisable to establish a family concept is primarily a marketing decision. That decision has an impact on the requirements for the products. Customers expect products belonging to a product family to exhibit more common features in terms of integration of product combinations or interface similarities in the case of technologically similar products. If the products do not adequately meet customer integration expectations, the family concept can have a negative market impact.

The term "product line" has gained a lot of attention, primarily in, but not restricted to the area of embedded software. We define:

Product Line = A set of products based on a common platform with defined (static or dynamic) variability tailored to different markets and users.

An example is an application software for offer calculation for craftsmen that needs to be adapted to any specialized craft it supports, like plumber, gardener, etc. Here adaptation means more than just customization. Adaptation means not just setting parameters differently, but applying changes to the code of the base product. That is why in this case we do not call the base product a product platform which by definition would have to be identical for all products based on it. Product lines are established for reasons induced by both business and technology management considerations.

All three concepts, the product platform, the product family, and the product line, increase the complexity of software product management. However, when used appropriately, they can accelerate development and marketing and increase return-of-investment.

2.2.2 Cloud Computing

One innovative area that has changed the IT landscape significantly is "cloud computing". The term came up around 2007, but the underlying concept is older. Since then, the concept has been extended into the mobile space. For this book, we will use the following definition:

Cloud Computing = Service and delivery model for the provisioning of IT components through the internet based on an architecture that enables a high level of scalability and reliability.

Cloud computing can be offered for

- IaaS: Infrastructure as a Service, e.g. processor capacity or storage,
- PaaS: Platform as a Service, i.e. infrastructure plus pre-installed enabling products, e.g. a database,
- SaaS: Software as a Service, i.e. infrastructure plus middleware plus application software, e.g. a customer relationship management (CRM) software.

Examples for IaaS are Amazon's Simple Storage Service (S3) and Elastic Compute Cloud (EC2) in the public cloud. Companies like IBM or HP help corporate IT organizations of larger enterprise customers to build private clouds that use internet technologies, but are strictly separate from the public internet.

IaaS providers have started to extend their IaaS offering into a PaaS offering that allows third-party developers to develop and offer their software products rapidly. Also governments have invested in such PaaS platforms. For example, the European Commission developed FIWARE, a platform which got adopted by a large number of startup companies already in its early years.

A well-known SaaS provider is Salesforce.com with its CRM application. Many interpret search engines like Google also as SaaS. This model means that the vendor does not sell a license to the customer, but a software-enabled service. The vendor will host, i.e. install and run the software on his own or a subcontracted computing center and take care of the installation of any updates. The customer gets the right to access the software via the internet and use it according to its contractual terms that will have the character of a Service Level Agreement (SLA). The customer stores his data on the vendor's system. SaaS promises unprecedented levels of scalability for customers without any significant investments in their infrastructure or operations. A lot of implementations follow a hybrid model where one software component runs on the user's device and communicates directly with the cloud component. The hybrid model is very popular with mobile app providers.

The SaaS delivery model may seem financially attractive. However, compared to a customer's on-premise cost of ownership, SaaS providers' real cost savings are confined to the savings gained from centralized management and infrastructure. These are higher if the software supports multi-tenancy which means the ability to run multiple clients in parallel on the same runtime instance of the software. Those savings are what the providers can share with their customers. Additional business benefits for a provider, especially in the consumer business, can come from opportunities based on the data that customers provide. The Google business model of offering targeted advertisements based on the customers' search profiles is a good example of this.

From a customer perspective, SaaS has several advantages:

- The customer does not have to deal with the operations aspects of installation, maintenance, data storage, and backup.
- The software is automatically on the latest release level (which is hopefully compatible with the previous release level).
- The customer usually perceives the new pricing models as advantageous compared to the traditional license model.
- In most jurisdictions, customers have a financial advantage since periodic payments for SaaS are handled as operating expenses whereas investments in infrastructure and software licenses are considered as capital expenses.

The disadvantages are:

- The data is stored under the control of the vendor. The customer needs to have sufficient trust in the vendor, and the vendor needs to ensure an appropriate level of security.
- The availability of the service is dependent on the availability of the internet that is typically neither controlled by the vendor nor the customer. The delivery of software services at the location of the customer is a business opportunity for telecom companies that they pursue under the name "mobile edge computing".

SaaS was preceded in the 1990s by the Application Service Provider (ASP) model. ASPs leased communication lines instead of using the internet. Most ASP applications were customer-specific whereas SaaS offers standard software in a multi-user mode that the individual customer can only customize to a limited extent. The lease-based single-tenant approach never gained significant momentum in the market. The trend towards SaaS was predicted by [RusKan03] as early as 2003 when the authors still called it e-service. According to market intelligence firm IDC, in 2020 more than 25% of software spending will be on cloud software services instead of a product license [IDC16]. The average annual growth rate will be 20%. These figures mean that the market share of cloud software services in terms of number of new customers will be significantly higher than these 20% since a lot of the product licenserevenue is still based on one-time charges.

2.3 Software as a Business

Consideration of software as a business implies consideration of the what and why of the software: the definition of the product and the markets, the delivery and sourcing of the product, and the competitive positioning. A software product manager who wants to define such a product strategy needs to have an excellent understanding of the software industry, its trends and drivers, and the relevant markets for the product.

What we have seen in the banking or music industries has only been the beginning. Software penetrates the core of more and more industries, be it software-based diagnosis and personalized care systems in the healthcare industry, software-controlled engines and brakes in the automotive industry, and even cars which no longer even need a driver. This importance of software is quite new to these industries. It requires not only technical but also cultural changes that can be facilitated by introducing software product management in these organizations.

At the same time, more and more IT components become commodity products, like storage, processors, or inexpensive or free software apps. For a company from any industry, the challenge is finding the right combination of low-cost commodity IT and differentiating non-commodity elements that improve the company's competitiveness. Enablers of competitiveness are the attractiveness of the company's products, the effectiveness and efficiency of the business processes of the company,

the abilities of the employees, and the design of the customer interface. It is a basic element of our economic system that these factors have been, are, and will be enabling differentiation in the market. Investments in IT that support these factors will be highly relevant for competitiveness in the future. Well known current examples of companies that are differentiating themselves in the market in exactly this way are Amazon, Apple, and Samsung. Their success is primarily based on innovative processes and customer interfaces that could only be implemented with IT. Commodity products can be part of those solutions, but they alone will not result in competitive advantage. Of course, when commoditization leads to lower prices for both hardware and software and thereby higher performance per $, then innovative offerings become financially feasible that were not feasible before.

2.3.1 Low Capital Investment

For software products, almost the entire product cost is incurred during development. As a consequence, a certain minimum revenue or minimum number of licenses must be achieved to reach the break-even point. In other words, software is one of very few products for which any additional revenue beyond the break-even point is almost pure profit. This characteristic of software products means that there is a high incentive—or necessity—for software companies to offer their products internationally to achieve the largest possible sales volumes. Internationalization usually requires translation of usage interfaces and documentation into the respective languages and addressing the specific requirements of each targeted country. The software market, in general, is international though there are some successful vendors for regional software solutions. This international orientation is an important element of the job of a product manager in software companies.

Compared to traditional industries that develop and produce physical goods, a software company needs relatively little capital and investments initially and in the course of its development. In principle, for software development "only" good know-how, a development environment, and a few PCs are required. Creation of software product deliverables is cheap because it can be done by duplication of media and printing of brochures and handbooks. Such production activities may not even be needed in case of internet distribution. In reality, even in the software industry it can be a bit more complicated, but the famous garage start-ups are not just myth. Anyone can start a software company—but not necessarily make it succeed. As is true for all startups, the secret of success is good know-how and professional management, with the difficult transition of company leaders from start-up entrepreneurs to businessmen. Eric Ries claims that "entrepreneurship is management" ([Ries11], p. 8). This sentence works both ways. Entrepreneurship means building an institution that automatically involves management ([Ries11], p. 15). On the other hand, a manager in an established company can benefit a lot from acting like an entrepreneur and applying ideas of Ries' "Lean Startup".

The low capital investment means low market entry barriers. Somewhere in this world, new software companies are founded daily—and die daily. The most

significant asset of software companies—human know-how—is in the brains of the employees, i.e. highly mobile. The fluidity of such know-how leads to very high innovation in the software industry. Technological innovations spread quickly and get introduced into new product versions. Many software products have ideas at the core of their value which may not be patentable, copyrightable, or intellectual property protected via "trade secret." This high speed of innovation and adoption lowers the technical entry barriers, i.e. the software market is characterized by low market entry barriers and extremely high competition and fluctuation compared to other industries.

In the IT industry, the hardware manufacturers are in constant competition. A manufacturer of goods fears nothing so much as commoditization, i.e. the ability of his customers easily to replace his products with those of his competitors. The end user is not faced with big functional or economic barriers if he wants to change for example, the brand of car he buys. Therefore, manufacturers of commodity products invest heavily in brand and image marketing. Also, they invest in bells and whistles that attempt to differentiate the product or make it to some degree proprietary (though the latter may backfire).

With software, competition is different. There is significant effort and risk involved in switching from an installed software product to a competitive product than for any other element of the IT infrastructure. This statement is valid for enterprise software, but also to some degree for consumer software. The main reason is that software is rarely used in an isolated way, but instead as an integral part of the complete software landscape, in an enterprise environment usually including self-developed applications. Unless there is a major quantifiable and visible-to-end-userimprovement in a replacement product, the IT professional will correctly reject any decision to replace a working product. He will receive zero credit if he succeeds (no visible benefit), and the wrath of all his end users if the product fails. Another aspect is the high level of company-specific customization. The degree of interdependence of a product with the other components of the software landscape is extremely high and very often not properly documented. That is why software once installed, integrated and used is very difficult to replace. It also means that ongoing product development, maintenance of older releases, and compatibility of new to old releases and versions have high importance and are a key success factor for software product management.

2.3.2 Law of Increasing Returns

As pointed out in [HoRoPL00], the software industry is governed by the law of increasing returns that was first described by the economist Brian Arthur [Arthur96]. This law says that a software product with a high market share will experience a further improvement of its market position just because of this high market share whereas a low market share leads to a further decrease. In other words, trends in market share intensify themselves. There are three main reasons for this

phenomenon: the network effect, the increasing cost of switching, and the trust in the market leader.

The network effect was described by Katz and Shapiro in [KatSha85]: "The utility that a user derives from the good depends on the number of other users who are on the same network." There are direct effects like standardization of interfaces that allow users of the same software product to exchange data and user experience more easily—they speak the same language. With users in different companies, this leads to a trend towards standardization of software products across companies. Indirect effects come from complementary products and services whose number increases when more customers use the base product (see [BuDiHe12], p. 21).

The increasing cost of switching characterizes how software "sticks" in a customer environment. The longer a user or company has a certain software product in use (e.g. text processing software), the larger the effort of switching to a different product. The data created with one program cannot necessarily be interpreted by the other program. Imagine the difficulties in a company that has changed from text processor "A" to "B." If, a year or two later, it is necessary to refer to or change a document created under "A" there must be a way to do that. So in some way the older program must either be kept or conversion provided for, which is a significant inhibitor to change. Also, the switch usually means significant education effort.

The trust in the market leader describes some of the behavior of customers when they make buying decisions. Consumers, as well as big companies, tend to rely on established brands and leading products. An established brand gives the investment the security that the chosen product will not disappear from the market the next day. Customers interpret the market leading position as a quality seal since many have already decided for the product. The standardization of interfaces and data is also relevant in this context.

The importance of market share for a software product is different dependent on the level of maturity of the market in which competition takes place. In an early technology phase, there are typically a large number of young products with similar functionality and the customers are in an orientation phase. The first products are tested; some early adopters make product decisions and invest in installations. In this phase, there is no market leader, but trends and camps emerge. The power and prestige of the respective camps will usually determine the market leader of the future. During the growth phase of a product market, two or at most three dominating products and vendors can be identified very quickly, i.e. within 12–18 months. In the maturity phase, one can observe all three of the effects described above. The combined effect is a consolidation and disappearance of the smaller vendors, and often a concentration on two or only one dominating product.

The law of increasing return makes market leadership more important for software products than for most other products. Of course, market leadership is an important competitive advantage for most products, be it for cost efficiency through economies of scale, be it as a marketing argument. But nowhere is market leadership so important for the future of a company in a market as in the software industry. That is why short term the goal of market leadership in a chosen market segment has priority over other company goals like revenue or profitability.

One note of caution on the law of increasing returns and the network effect in particular: they alone do not guarantee sustained success when new technologies emerge. For example, Lotus dominated the spreadsheet market with "1-2-3," Netscape had a majority share of the browser market, and WordStar dominated the text editing market. A competitive bundle appeared, Microsoft bundling or selling with Windows, and broke the domination. The disruption happened even though the market was mature and the product well established. A vendor must be vigilant, particularly as new technologies are introduced. The best strategy appears to be continuous evolution and improvement and never resting on your laurels. An example is Adobe's Photoshop. Although Adobe's high price strategy created a price umbrella for others to come in under and compete with compatible or similar products, the product continued to be successful for a long time. Nevertheless, Adobe made a radical switch to a SaaS model for most of their products.

New software technologies develop and drive new markets for software products. These markets show fast, often drastic growth, during which a high number of competitors boil down into a few winners and lots of losers. Software technologies and their markets can be compared to layers of an onion. New layers are created continuously, and the old ones dry out and die. The art of running a software company is to leave the drying outer layers in time and jump onto a juicy inner layer, a new market in which there is a new fight for market leadership.

2.3.3 The Financial Life Cycle of a Software Product

As just discussed, the start-up capital requirements for a software company are modest. If one has a good idea, and perhaps the beginnings of a demo or prototype, venture capital can be found which is sufficient to begin. Still, just as in almost any startup, there will be a great deal of expenditure before the first dollar of revenue is earned for a software product. Software is unique in that most of the capital outlay occurs during this pre-revenue period. As soon as revenue is earned, excepting major enhancements, the software product incurs small variable cost and high marginal profit. With SaaS, variable cost is a bit higher than a one-time electronic distribution of the software due to hosting.

To earn revenue with a software product, two common revenue models compete with each other: the one-time charge (OTC) model and the periodic charge model. Once sales begin, there will be pressure to show profits, certainly pressure to show dramatic yearly increases in revenues. In a periodic charge model with recurring charges, there is no acceleration of revenue through initial sales. Revenue is earned over time as the product is used. This revenue model contrasts with the up-front revenue model that is based on the sale of a one-time charge (OTC) perpetual license. Like someone who owns an apartment building and rents out the apartments, a software company with attractive products who licenses them using recurring charges may make a great deal of money in the long term. However, deferred recurring revenue is not the way to show the most dramatic financial growth as one might want to do in a start-up, particularly if one were contemplating

an Initial Public Offering (IPO). This communication problem may influence a company to offer its licenses on an OTC basis, relying on maintenance charges to provide a recurring stream of revenue following the initial sale. With SaaS, periodic charge models have turned into the standard.

As the product becomes established, the objective shifts to wanting to maintain price and preserve revenue stream with less regard for quick growth. Achieving this shifted objective requires clarity of strategy, strength of will, and discipline in terms of discounting, periodic enhancements, continued marketing to increase share. As we said in the previous section, there is no fear as great as that of commoditization during this phase. Most software businesses will need the cash being generated by their successful products to finance the development of enhancements and new products, as well as potentially the acquisition of companies with interesting and innovative products. So the objective in the medium term is often to prolong product life as much as possible with modest expenditures to maximize profits, yet retain market leadership.

As companies grow, they often choose to grow by acquisition, particularly if they have cash. One can see this clearly with IBM, Computer Associates, Oracle, and others. IBM has been acquiring software companies at an increasing rate of more than one per month for several years. Reasons can be to buy innovation and shorten the time-to-market, to buy market share, or to get specific skills fast [Popp13].

Eventually most products move into a sunset phase where they reach their end of life, replaced by a new product in the vendor's portfolio (hopefully) or perhaps by a competitor. However, a product may be devilishly difficult to replace if it is tightly integrated into a company's infrastructure. Therefore, it behooves the vendor to provide an easy, effective, and financially attractive migration to his replacement product. If the migration to the vendor's follow-on product is difficult or expensive, the sunset will invite the customer to consider competitive alternatives. For customers who have a perpetual license, it is even harder to get them to migrate to a new product. Microsoft and their challenge to motivate customers to migrate to newer versions of Windows or Office is a good example. Microsoft tried to address this issue by offering Windows 7 and 8 users in the consumer market a free migration to Windows 10 for the first year following availability.

2.4 Business Models

A business model describes the "rationale of how an organization creates, delivers and captures value by interacting with suppliers, customers and partners" ([OstPign10], p. 14). The business model reflects how a company intends to make money. The term "business model" has become popular with the Internet, which has opened up a wealth of possibilities to do business. The term can be applied to any business organization though it is often considered at the corporate or business unit level. A business model analysis can be helpful for a software product manager when analyzing potential partners. While relevant for an individual product, its

consideration can also make sense for a solution that spans multiple products and services. We will come back to this in the chapter on Product Strategy.

2.4.1 Describing a Business Model

A Business Model is a conceptual model that describes the products and services considered and how revenue streams relate to these products and services. Teece provides some scientific foundations to the concept [Teece10]. Business models can be classified according to three dimensions: "the type of products or services provided, the business model archetype, and a revenue model" [PoppMey10]. A business model is constructed by choosing one or more combinations of the three elements of a business model (see Fig. 2.1).

According to [PoppMey10], types of products or services can be

- Financial products (cash and other assets),
- Physical products (real, physical products, durable and non-durable goods),
- Intangible products (software but also other intellectual property, knowledge and brand image), and
- Human services (people's time and effort).

Business Model Archetypes are basic patterns of doing business. Available archetypes are creator, distributor, lessor, and broker.

- A creator uses supplied goods and internal assets and transforms them to create a product sold to customers. It is important to know that the main work done by the creator is designing the product. An example is Apple. Apple designs the iPod in California. So Apple is a creator.

Business Model Matrix		Type of Products and Services			
		Financial	Physical	Intangible	Human
Business Model Archetype	Creator	Entrepreneur	Manufacturer	Inventor, Developer, Author	./.
	Distributor	Financial Trader	Wholesaler, Retailer	IP Distributor	./.
	Lessor	Financial Lessor	Physical Lessor	IP Lessor	Contractor
	Broker	Financial Broker	Financial Lessor	IP Broker	HR Broker

Fig. 2.1 Business model matrix [PoppMey10]

- A distributor buys a product and provides the same product to customers. Obvious examples are companies in the wholesale and retail industries, like Sears or Saks, or Apple's online retail store for applications, the iTunes AppStore.
- A lessor provides the temporary right to use, but not own, a product or service to customers. Examples are landlords, lenders of money, consultants and software companies that license their software to customers. For human services, human resource lessors lend their employees' time to customers.
- A broker facilitates the matching of potential buyers and sellers. A broker never takes ownership of the products and services. An example is a stock broker. Another example is Google's advertising business, which matches the advertiser with potential customers.

Single products and services can be offered stand-alone, or they can be offered as a bundle, which is a combination of products and services. Software vendors are typically offering intangible products and act as a creator and lessor of software, but also offer human services like consulting, maintenance and support.

A revenue model describes how revenue is generated from the product offering. Details will be discussed under Pricing (Sect. 3.10).

One of the widespread tools for reviewing and evaluating existing business models is the Business Model Canvas. It is also used to systematically invent new ones that change the way a product competes. The Business Model Canvas consists of nine segments, see Fig. 2.2. On the right-hand side, we find the segments representing market- and customer-facing views (e.g. customer segments and revenue streams). On the left-hand-side, we find those representing company-internal and supplier-facing views (e.g. key resources and cost structures). In the middle, the value proposition links the two sides or views.

Each of the dimensions of a business model leads to particular entries in the Business Model canvas. The entries are as follows:

- Products, services, or bundles are compensated by the customer. Compensation can be monetary (revenue streams) or non-monetary (exchange of products, services or information). So for each product there always is compensation, which in many cases is a revenue stream.

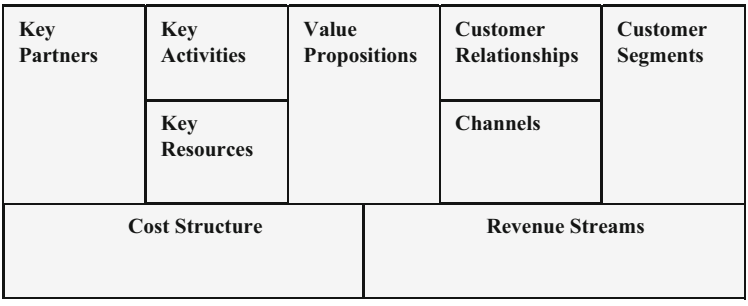

Fig. 2.2 Business Model Canvas [OstPign10]

- Products, services, or bundles have a corresponding value proposition.
- For each type of product, service, and bundle there is a specific cost structure.
- Business models are executed by activities, some of them being key activities. Activities are carried out by resources, some of them being key resources.

An example will be given in the next section.

2.4.2 Business Models in the Software Industry: Software Product Company

The prototypical type of company in the software industry is the software product company, also known as software vendor, that develops, offers, and maintains software products. The customer buys a license, and installs and runs the software on-premise or as a cloud-based offering. To do the latter, the customer signs a contract for a cloud-based service where the vendor runs the software, and the customer accesses the service over the internet.

Figures 2.3 and 2.4 give a description of a software vendor's business model. We assume that the vendor has both on-premise and Cloud-based businesses (which requires hosting, shown in Fig. 2.3 as "Physical Lessor"). The vendor develops software, distributes it directly and/or indirectly, and sells licenses, i.e. act as a lessor. Also, the vendor offers product-related human services like maintenance, shown in Fig. 2.3 as "Contractor". Figure 2.3 demonstrates that the vendor covers five elements of the matrix. The business model canvas (Fig. 2.4) shows more details of the business model.

The business model of a software vendor assumes standard software products that are suitable for a large number of customers.

The sales of a software product can be combined with product-related services that can be priced based on actual effort, or as a package with the software license,

Business Model Matrix Software Vendor		Type of Products and Services			
		Financial	Physical	Intangible	Human
Business Model Archetype	Creator	Entrepreneur	Manufacturer	Inventor, Developer, Author	./.
	Distributor	Financial Trader	Wholesaler, Retailer	IP Distributor	./.
	Lessor	Financial Lessor	Physical Lessor	IP Lessor	Contractor
	Broker	Financial Broker	Financial Lessor	IP Broker	HR Broker

Fig. 2.3 Software Vendor's business model matrix

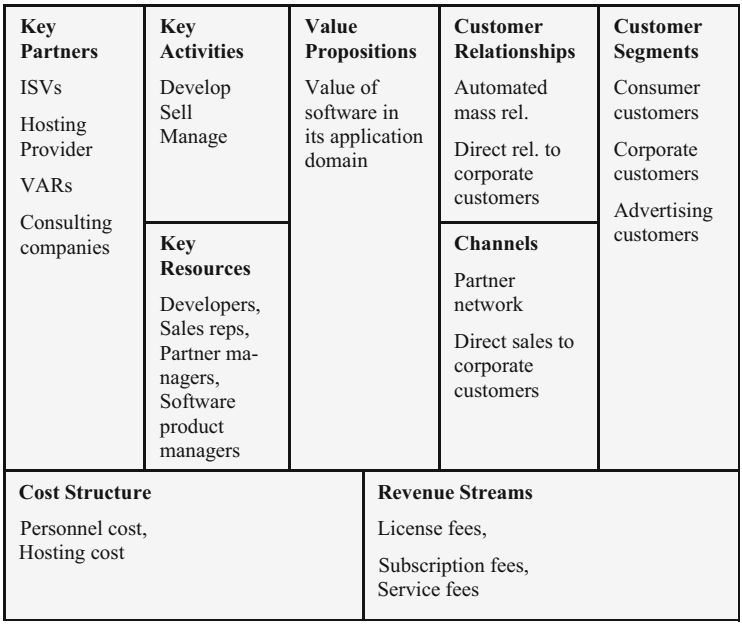

Fig. 2.4 Software Vendor'sBusiness Model Canvas (example)

or as a subscription as is customary for product maintenance. According to Cusumano [Cusuma03]: "A general rule of thumb is that, over the lifetime of using a software product, enterprise customers pay between one and two dollars in service and maintenance fees per dollar of software license fees (the up-front product cost)." In particular when the economy is bad and product sales are down, the importance of this income for the survival of a software vendor becomes obvious. That is why maintenance fees and even more so the pricing model of Monthly License Charge (MLC) are so attractive. There is no large up-front payment, but smaller monthly payments. This pricing model has been a pleasant pillar for IBM for a long time since it resulted in a more balanced, smoother revenue stream. The stock market deplores nothing as much as unpredictable earnings. With SaaS, the recurring pricing model has returned under the name 'subscription fee'.

2.4.3 Business Models in the Software Industry: Professional Service vs. Product

While both the software product and the professional services business models are attractive, there are significant differences that make them difficult to combine. In this context, it is important to differentiate between the different meanings of the term "service" that we introduced in Chap. 1. There is a continuum from services to products that can be confusing. It is shown in Fig. 2.5.

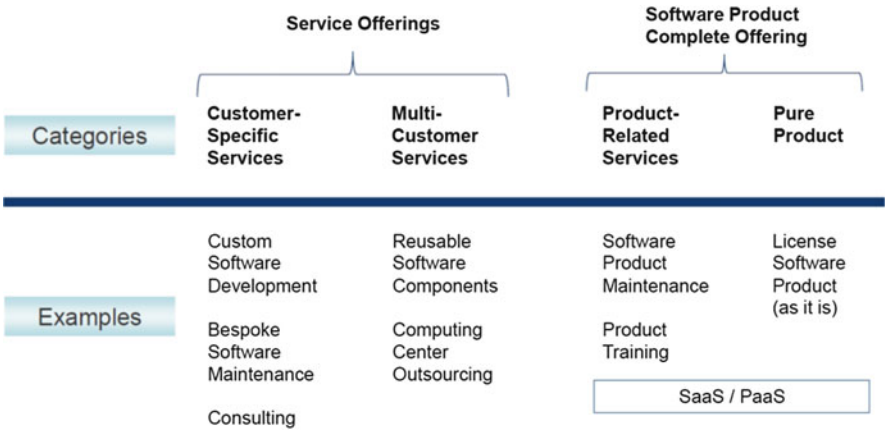

Fig. 2.5 Service-product continuum

Of course, a company can follow the business model of a professional service company that develops customer-specific software. This business model means that the development of software is paid for, either based on effort or at a fixed price. The focus is not on a software product, but on projects and the employees who deliver the service. The goal is to implement an application, according to the requirements of the customer. It is typically not designed for a broad and flexible spectrum of usage scenarios.

The "service" in SaaS or PaaS is a technical service. For these offerings, it makes more sense to look at them as product offerings. Directly product-related services like maintenance or training ought to be managed as part of the whole software product offering. The term "whole product" was popularized by Geoffrey Moore [Moore14]. It means the combination of the base product and additional products and services by the same or different companies that together provide a convincing solution to customers in the target market.

In contrast, customer-specific human services like custom software development constitute a professional service business. This differentiation is required because service (or project) and product businesses have different business models as shown in Fig. 2.6.

In most cases, the financial characteristics of software products are opposed to those of professional services. Software requires high upfront investment and has low revenue-dependent variable cost. Once the original investment is recouped, additional sales result in very high profit margins. Professional services, in contrast, require low upfront investment and have high revenue-dependent variable cost. Of course, once a services team is hired the personnel costs can be considered as fixed. It would be difficult to lay off staff when there is not work to keep the employees busy and rehire the same skill pool when there is more work. Pricing is fundamentally different. For professional services the standard pricing approach is cost-based, whereas for software products it is value-based (see Sect. 3.10). Key performance indicators (KPIs) are different as well. For services, companies look at utilization rate, i.e. the percentage of work time of service employees that is paid for by customers. If that KPI is applied to a product business that business is killed

	Service Business	Product Business
Focus	Customer, Project	Market, Product
Financial model	Small investment, Low risk, Continuous moderate profit	Significant upfront investment, Higher risk, Potentially high profit
Price calculation	Cost-based (cost + margin), Existing software usually included	Value-based, Software product usually priced separately from services
KPI	Utilization, Average daily rate	Market share, Profit
Market evaluation	Moderate	Much higher

Fig. 2.6 Differences between (professional) service and product business

because the required upfront investment would never be made. The reason why a lot of service companies want to start a product business is the market evaluation of the company. For a service business, it tends to be in the range of 2–3 times the annual revenue, for a product business often in the range of 10.

Software product management has been common practice for most software vendors. However, this does not mean that all software vendors manage their products explicitly. Often start-up companies are founded with the intention to go to market as software vendors, but lack the capital needed to develop the first release. If venture capital or bank credits are not available, an alternative is driving development through customer-specific projects that the customer pays for. Customers can often be found on the basis of sufficiently attractive prototypes.

An interested customer may be willing to pay for the way from a prototype to a commercially usable product that meets his specific requirements. However, he will be watchful that his money is not "misused" for the way to a more widely usable software product. That is why this approach does not succeed in most cases and often leads to separate code bases for each customer and not to a more widely usable product. As long as further development and maintenance of these code bases is paid for by the respective customers according to effort, the model can work. But it is the business model of a service company, not of a software vendor. When company and customers switch to the software vendor model, the customers only pay standard maintenance fees that would be sufficient for maintaining a common product code base, but not separate customer-specific code bases. A mix of customer-specific development and standard maintenance frequently results in economic problems for the vendor.

Our experience suggests the following rules of thumb:

- The relation of the effort for a prototype versus a piece of software commercially usable in one customer environment is 1:3.
- The relation of effort for a software commercially usable in one customer environment versus a software product that can be used by a higher number of customers is again 1:3.

What can we conclude from this? The executive management of a company must be clear in what the company is supposed to be, i.e. which business model it intends to follow. Michael Cusumano, professor at MIT Sloan School of Management, writes in [Cusuma03]: "Regardless of the balance of products and services they choose, managers of software companies must understand what their primary business is, and recognize how the two differ—for selling products requires very different organizational capabilities than selling (professional) services." The cultural differences between software and service companies often lead to failure whenever a service company tries to turn a piece of software that had been developed based on a service contract into a standard product. However, this cultural change is possible. An example is the CAD software CATIA developed by Dassault in France that became a very successful software product.

These are the reasons why all major IT companies that have both product and service business used to separate them strictly on the executive management level. A good example is IBM, who tried to establish a PaaS/SaaS business run out of IBM's service division that had a lot of outsourcing experience. It did not work too well. So in 2014, IBM relaunched its PaaS/SaaS business under the name Bluemix managed by IBM's product group. With their 2015 reorganization, IBM left that proven path of separating product and service business by establishing business units like Analytics or Commerce that combine software products, additional partner products, and product-related services. This experiment bears a lot of risk.

2.4.4 Business Models in the Software Industry: Open Source

Usually, ownership of software code is with the company that develops it. In case of custom software development for an individual customer, it is a matter of contract negotiation whether ownership is with the customer, the professional service company or both. We cannot recommend strongly enough that thius be crystal clear in the contractual wording in order to avoid later conflicts over intellectual property, royalties, right to market, etc.

With the emergence of open-source software like the Linux operating system, an alternative to the product ownership paradigm has proven its feasibility. A growing community of developers contributes in a self-managed and parallel way to the development of a complex system that has gone through the metamorphosis from a cult object to a de facto industry standard. Through the internet, the Linux code is available to everybody at no cost whereas the packages of Linux distributors that add installation and implementation tools and documentation to the code have a price tag.

Given the success of the open-source movement with software like Linux, Apache, the middleware JBOSS, or the MySQL database, there is some speculation that all software may be free of charge over time. We think this extrapolation is misleading. The history of open source shows that motivations can be observed that are clearly rooted in specific characteristics of the contributing programmer community. There is a strong anarchic element that favors the elegance and beauty of a

technical solution versus any economical considerations. This element tends to rebel against restrictions that companies put on their employees or that the economic system puts on its market participants. A key factor of the open-source community has always been resistance against companies dominating the market, in former times IBM, then Microsoft. In the community, recognition by peers, i.e. other acknowledged specialists, is more important than economic success. However, many of these programmers use the recognition for getting well-paid jobs. The contradiction between the economic interest of the individual whose work as a programmer has to pay the bills and the anti-economic attitude of the community characterize the movement. Since this community consists exclusively of "techies", i.e. technically oriented programmers, it does not surprise that the primary focus has been on infrastructure software, i.e. operating systems and middleware. In recent years, there has been an increasing number of application-oriented open source projects, but none has reached market share comparable to the open source flagships like Linux or Apache. Applications can only be developed in cooperation with specialists from the application domain who typically do not share the anti-economic attitude of the open-source community. Over time, some companies have embraced the open source model for their own commercial reasons, e.g. Netscape [Hecker99]. Other companies work with hybrid models by combining proprietory and open source software [BoGiRo06]. Demil and Lecocq [DemLec06] look at open source ecosystems as controlled by "bazaar governance".

Of course, commercial software can make use of open source code, but vendors need to beware of legal implications and risks (see Sect. 3.6). Once a piece of open-source software has gained a certain market share, it becomes interesting for commercial software companies. Based on the open-source code, software products can be sold with packaging and product-related services justifying the price tag. Examples are companies like Red Hat or Suse (acquired by Novell) in the Linux area. Established vendors have also embraced open source. Examples are Sun's acquisition of MySQL (now Oracle), or IBM that has jumped on the Linux bandwagon. What is the motivation behind this? The effort of a software company for the continuous development of complex software like an operating system is enormous. For this to make sense for the bottom-line, either the product price has to be high—an example is IBM's DB2 for z/OS—or the volume has to be high—as for Microsoft's Windows. A relevant open-source operating system, which is offered at almost no cost, may encuorage a manufacturer to support that open source project, while making sure that his hardware will continue to be supported. Such cooperation may be financially more attractive than a self-developed software product.

Apps in app stores can usually not be considered as open source software even if their price is zero. They are usually developed by individual developers, not by a community, and their code is not made available to the public. Also, the developer will often try to generate revenue in other ways described below.

2.4.5 Business Models in the Software Industry: Free Commercial Products

Besides open source, there are a lot of commercial products that do not cost anything or very little. They can usually be downloaded or used on the internet. Low-cost pricing is typically driven by commercial interests and marketing strategies. The following variants are known: freeware, trial-ware, or upgrade-ware.

Freeware is the offering of a product at no charge. Examples are viewer or player products like Adobe's Acrobat Reader and Flash Player or Real Audio's player. Here, the software vendor intends to make the usage and exchange of files in "his" format easy and at the same time to create demand for his costly full-function-product that is needed to create the files.

Trial-ware is the offering of a product either as a web download or sometimes media (CD or DVD) which offers a trial period after which it requires payment of a fee for continued use. Demo versions often have restrictions in functionality or capacity or become unusable after the trial period. This type of software does have the character of products, is part of a product family and is managed by the respective product management organization. A plan is needed that specify the functions to be made available and for how long they should be available since this product behavior has to be implemented by Development.

Upgrade-ware (or Freemium) is the offering of a product with a base version at no charge and with enhancements that require payment. The user is encouraged to upgrade to obtain additional function. Base versions may range from full function products, where the upgrades offer additional bells and whistles, to limited (or no) function programs that simply tell you what they would do if they were fully functional. Especially with online games, this business model is also known as the freemium model where users can play a game for free, but as soon as they want extra features, they need to pay. A variant of this is Adware where the software can be used for free, but contains advertisement. If the user wants to get rid of the ads he needs to pay a fee.

SaaS may be offered for free and be financed through advertising. Examples are search engines like Google or Yahoo and Google Apps, the office and collaboration suite that Google has launched. Google Apps can be seen as Google's attempt to transfer its extremely successful search business model that is based on advertising revenues to software market segments whose business models have traditionally been based on license fees. Social media platforms like Facebook are also trying to monetize their reach through advertisement. From a customer perspective, these offerings do not require monetary payment. However, there is a price to be paid in terms of being exposed to ads and revealing information about one's identity or interests. An increasing number of apps in the app stores of Apple or Google also fall into this category, i.e. the app developers try to generate revenue through ads.

Both open source and SaaS offerings providing revenue through the support of paid advertisements are examples of Wikinomics at work. The term "Wikinomics" was coined by Don Tapscott and described in his book of the same name [TapWil06]. It stands for the emergence of new business models that are based

on the network effect enabled by the internet. We can expect to see more innovative business models over time in this context, in particular in connection with social and business communities on the net, some of which will impact software vendors.

In addition to these business models of software vendors and internet companies, there are other business models in software ecosystems. Examples are Value-Added Resellers (VARs), System Integrators, Hosting Providers, etc. (see [Kittlaus14]). We will come back to these ecosystem-oriented business models when we discuss software ecosystems in Sect. 3.11.

In software-intensive industries, the business models are governed by their respective products. These are not software business models. However, given the increasing importance of software in the products of almost all industries, software influences these business models. Software may even lead to disruptive business model revolutions, as in the music industry or the newspaper industry.

2.4.6 Business Considerations for Corporate IT Organizations

Corporate IT organizations have traditionally not had the view that they developed, marketed and sold software products. Typically they saw themselves as the master and caretaker of an IT system that consisted of hardware and an applications landscape that was difficult to comprehend and manage because it was most often not thoroughly documented. Where and when a new function was implemented was often more a case of arbitrary organizational responsibilities and informal relationships than of a forward-looking planning of individual applications or the complete landscape. This randomness of decision-making was typically accompanied by severe communication problems between IT and the business units. The increasing dissatisfaction of the internal contract givers with the price and performance of IT—whether justified or not—has led to initiatives over the last 30 years aiming at making the IT organizations more manageable.

One area has been in providing more "customer" control, replacing the former dissatisfying fixed cost allocation with a more cause-oriented, sometimes transaction-based cost allocation. Also, there have been attempts to destroy the former monopoly of the IT organization by introducing competition. In some companies, the business units were allowed to order hardware and software without coordinating with the IT organization. This competition succeeded in weakening the IT organization, but unfortunately it also weakened the whole company because the inexperience of the business units sometimes led to poor decisions or was taken advantage of by the vendors. This situation resulted in an unmanageable collection of badly integrated islands with increasing data redundancy and risk of data loss. We are currently seeing a very similar trend with SaaS offerings. Since the pricing of SaaS allows business units to book the cost as operational expenses, they can easily work with SaaS without involving the IT departments unless governance rules forbid this.

The idea of introducing competition was also the main corporate motivator for outsourcing IT organizations into independent companies that were still fully

owned by the parent corporation. The unfortunate executives of these companies were expected to achieve the square of the circle: strengthen customer orientation, lower cost, and develop new business as a service provider on the open market. It has hardly ever worked. Customer focus suffered because established communication paths were disrupted by the new organizational and often physical separation. The cost structures could only be improved long term. In fact, since many of these separated IT companies were expected to turn a profit and executive incentives were established to promote that, the companies added a profit margin to the fees they charged their largest customer, their parent, thereby increasing corporate costs for IT. This in turn provided additional incentive for business units to buy their own small computers or outsource their work to outside service organizations with more aggressive pricing. In Europe, the work councils made sure that at least short term no employee suffered due to the outsourcing. In the US, social pressures and potential loss of government incentives often had similar effects. And the new business did not materialize because selling on the open market was foreign to the company's culture and not realistic as any potential customer knew that he would always have second priority versus the parent corporation. Another issue was the lack of focus of the outsourcer on his client and in the welfare of the client's business.

Then there was a trend of external outsourcing to big IT service providers that imposed their outsourcing management and processes and often took over a large number of employees. Outsourcing a data center can lead to significant cost savings if the vendor passes on the cost savings realized by the consolidation. For software development, this is rare. Even if the service provider can develop more efficiently and at a lower cost, his profit orientation will jeopardize the benefits for the host company. Also, a formal requirements management process will limit the service provider's ability to cooperate flexibly. Only if additional customers for the application landscape can be won, significant cost advantages can be enjoyed. Successful examples are joint IT service providers of major German banking groups that provide services to a higher number of banking institutions, or service companies like First Data in the US that achieve significant economies of scale.

So far, attempts to make internal IT more manageable have led to mixed results. There is one positive outcome: The executives of the IT organizations—whether outsourced or not—have gained a much stronger entrepreneurial view of their organizations. Business units are today considered more as customers than as solicitors. The willingness to understand and manage the offerings as products and services has increased. This product and service orientation has made these IT organizations similar to software vendors, with the result that software product management has started to be accepted as a useful concept by the corporate IT organizations as well. The core elements of SPM are the same for software vendors and corporate IT organizations. The priorities may differ.

The increasing pervasiveness of IT and in particular software across most industries is leading to a changing view of high-ranking executives. This change is visible both in corporate IT organizations responsible for business infrastructure and in software development units that produce software embedded in the

company's products and services. For these organizations it is challenging to meet the increasing expectations in terms of value contribution, time-to-market, and innovativeness. Software product management can help a lot in these environments to improve communication and cooperation, and to focus more on business value.

2.5 The Software Product Management Framework

Given the broad spectrum of tasks and responsibilities of a software product manager, some structure is welcome. The first attempt to provide structure is the Reference Framework for Software Product Management developed by Inge van de Weerd, Willem Bekkers, Sjaak Brinkkemper, and colleagues at the University of Utrecht, Netherlands in 2006 ([WBNVB06], see Fig. 2.7). It focuses on the core activities of a software product manager in the areas of product planning and includes recommendations for portfolio management, product roadmapping, release planning, and requirements management.

The framework's structure is based on the entity hierarchy portfolio, product, release, and requirement that are managed with a set of activities. These activities are performed by a product manager. The activities belong to the core activities in the ISPMA framework. When interpreted as a process, the Utrecht framework shows the core of what some software vendors call their product life cycle management (PLM) process which is an iterative process for software repeated for every release.

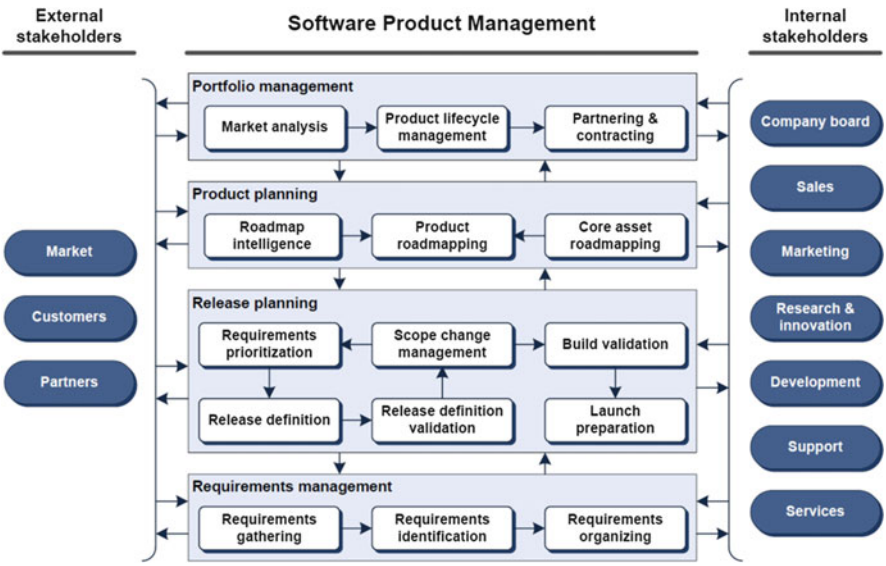

Fig. 2.7 Reference framework for software product management [WBNVB06]

We consider this framework as very helpful and shall come back to it when we discuss the individual activities. However, it does not cover all tasks of a software product manager. For example, economic aspects such as the business case are only indirectly reflected in this framework. In our view, the business aspects must play a much more prominent role in a software product management framework.

Bekkers e.a. published a detailed specification of the framework in [BVSB10], which they had developed and evaluated by studying a large number of small and mid-size enterprises (SMEs). The focus is on ways how to improve practices. In Fig. 2.8, "→" points to the practice of next-higher maturity, hence what the next practice is that should be adopted for increasing maturity.

Practices	**Improvement Sequence**
Portfolio management	Market analysis: trend identification → market planning → customer win/loss analysis → competitor analysis Partnering & contracting: service level agreements → intellectual property management → distribution channels reviews → pricing model reviews Product lifecycle management: product lifecycle analysis → portfolio innovation → portfolio scope analysis → business case reviews → product lines
Product roadmapping	Roadmap intelligence: product analysis → society, technology, and competition trend reviews → partner roadmaps Core asset roadmapping: asset identification and registration → make or buy decisions → core asset roadmapping Product roadmapping: short-term roadmap → internal consultation → theme identification → long-term roadmap → external roadmap
Release planning	Requirements prioritization: internal stakeholder involvement → structured prioritization → customer involvement → cost-revenue consideration → partner involvement Release definition: capacity consideration → standardized release definitions → internal communication → optimized requirements selection → multi-release definition Release validation: internal validation → formal approval → business case definition Change management: change requests → milestone monitoring → impact analysis → scope re-planning Build validation: internal validation → external validation → certification Launch preparation: internal communication → formal approval → external communication → training → launch impact analysis → update external documentation
Requirements management	Requirement elicitation: passive basic registration → centralization → automation → proactive internal stakeholder involvement → proactive customer and partner involvement Requirements definition: uniform documentation → requirements validation → grouping by similarity → automation Requirements structuring: requirements typing → requirements lifecycle management → requirements dependency linking

Fig. 2.8 Recommended improvements to product planning practices [BVSB10]

Other frameworks consider such business aspects. One is the Pragmatic Marketing® Framework [PragMark16]. It mixes product management and product marketing aspects, however. Kittlaus and Clough [KittClou09] defined the roles of product manager and product marketing manager as separate and focused only on the software product manager in their SPM Framework. Also Ebert [Ebert07] focused on product management when he proposed how to manage a software product along the product's lifecycle phases of strategy definition, concept development, market entry and development, and evolution.

The ISPMA framework [ISPMA16] (Fig. 2.9) is an integration and consolidation of the three frameworks from Utrecht, Kittlaus, and Ebert. The International Software Product Management Association (ISPMA) facilitated a consensus between these academic and industrial opinion leaders [Fricker12]. In contrast to other reference models, the scope and contents of the ISPMA framework continues to evolve by being discussed and adjusted by the product management experts that are members of the ISPMA knowledge network. It hence adapts to the evolving understanding of the discipline.

The ISPMA SPM framework provides a holistic view of the elements of software product management. The framework can be used as a model to establish and improve the discipline of software product management in an organization. The structure of the framework is as follows:

- The horizontal structure (columns) follows the functional areas of a software organization.
- Vertically within the columns, the structure is based on a top-down approach, i.e. from strategic and long-term to operational and short-term. However, the

Strategic Management	Product Strategy	Product Planning	Development	Marketing	Sales and Distribution	Service and Support
Corporate Strategy	Positioning and Product Definition	Product Life-Cycle Management	Engineering Management	Marketing Planning	Sales Planning	Service Planning and Preparation
Portfolio Management	Delivery model and Service Strategy	Roadmapping	Project Management	Customer Analysis	Channel Preparation	Service Provisioning
Innovation Management	Sourcing	Release Planning	Project Requirements Engineering	Opportunity Management	Customer Relationship Management	Technical Support
Resource Management	Business Case and Costing	Product Requirements Engineering	User Experience Design	Marketing Mix Optimization	Operational Sales	Marketing Support
Market Analysis	Pricing		Quality Management	Product Launches	Operational Distribution	Sales Support
Product Analysis	Ecosystem Management			Operational Marketing		
	Legal and IPR Management					
	Performance and Risk Management					
Participation	**Core SPM**		**Orchestration**			

Fig. 2.9 ISPMA SPM framework V.1.2 [ISPMA16]

interdependencies of the elements within each column (and also across columns) are more complex than can be fully expressed in a two-dimensional structure. There are some cases where the actual doing requires iterative processes that go back and forth between elements until everything is synchronized. A good example is the Product Strategy column where this kind of iterative approach is mandatory between most elements before a product manager gets to a consistent strategy. Also, the elements "Ecosystem Management" and "CRM" contain both long-term and short-term aspects.

- There is an additional overlay structure with "Core SPM", "Participation", and "Orchestration". In large companies, corporate functions are typically responsible for market analysis and product analysis together with the participating product manager. In small companies, the product manager may be responsible alone. In both cases, the elicitation of reliable information on market and product on a frequent basis is part of the core SPM responsibilities.

Typically, software product managers have direct responsibility for the activities marked as "Core SPM", in particular, "Product Strategy" and "Product Planning" [MagNiSmoFri17]. For the activities under "Strategic Management", software product managers participate by representing their products on the corporate level. For example, in portfolio management, product management provides input and makes use of the results. For the activities under "Development", "Marketing", "Sales and Distribution", and "Service and Support", the direct responsibility lies typically with other units in the company. A software product manager has to orchestrate these activities so that they are performed in line with product strategy and plan. Given the broad set of responsibilities, prioritization is needed on an ongoing basis and can be based on the estimated impact of prioritization decisions on short and long-term profitability.

2.5.1 The Four Software Product Scenarios

While we consider the role definition of software product manager described above as universal, its implementation and priorities depend on the type of product a software organization offers. Here we see increasing heterogeneity. We look at four scenarios for software vendors.

Figure 2.10 shows a classification by using two types of runtime environments for software product instances and two software product life cycle phases.

Vendor-controlled means that the software vendor decides which changes are made when in the runtime environment. Vendor control is typical for rather unregulated environments like business-to-consumer (B2C) internet platforms and SaaS or B2C license products that offer automated maintenance over the internet. Customer-controlled means that customers want to supervise the runtime environment, often because of quality, security, or regulatory concerns. Customer control is typical for business-to-business (B2B) software license products, but can

Software Product Scenarios		Life Cycle Phase	
		New Product Development	Existing Product Evolution
Runtime Environment	Vendor-Controlled	Powerboat	Speedboat
	Customer-Controlled	Icebreaker	Cruise Ship

Fig. 2.10 Software product scenarios for software vendors [Kittlaus15]

also be found with B2B SaaS, in particular when tied to business process outsourcing.

The classifications also differentiate between the initial development of a product and the evolution of an existing product which already has customers. With new product development, there is a high level of uncertainty and risk. The focus is on releasing a minimum viable product as fast as possible. Once the product is rolled out, the focus shifts to extending the product scope and target market. Also, compatibility and migration aspects become relevant. We will take a more detailed look at the later phases of the product life cycle in Sect. 4.4.

In particular, release planning and product requirements engineering in the SPM Framework's Product Planning column work differently in the four resulting scenarios:

Powerboat SPM focuses here on defining the minimum viable product for the first customers. This goal requires a close link between product management and development, where the product manager often assumes the product owner role (in Scrum terminology), and extensive prototyping. In parallel SPM needs to work with potential customers on contents and business model, and with Marketing on positioning and pricing. Investments need to be justified based on a strategic perspective. The product vision (aggressive), the business model (one-page canvas), the product strategy (very high-level), the roadmap (high-level), and the business plan (aggressive) play an important role. Release planning and requirements engineering are performed in support of short-term experimentation, which some call "pivoting". A product example is a new internet e-commerce platform before its initial launch.

Speedboat SPM focuses here on extending the product scope and thereby increasing the target market. These goals require ongoing analysis of the actual usage of the product, of the market, and of competition. Depending on the organization's size, SPM and product owner roles may be separated, but remain closely linked. Product strategy and high-level roadmapping become important in combination with life cycle management. Release planning focuses on prioritization of

individual requirements and scoping of development iterations. Requirements engineering is a mix of analysis and experimentation through customer discovery. If the organization does not implement governance functions like Architecture, things can become messy very quickly. Governance, compatibility, and migration tend to decelerate the velocity of the organization compared to the Powerboat phase. A product example is Google Docs.

Icebreaker SPM focuses here on defining the minimum viable product for the first customers. In the customer-controlled context, this goal requires extensive domain analysis with potential customers as a basis for requirements engineering and planning of the first release with a special emphasis on regulatory requirements. If a pilot customer is involved, a major SPM task is making sure that requirements are sufficiently generalized so that the first release does not become overly customer-specific. The interface between SPM and Development depends on the chosen development methodology. Product strategy and roadmap already need some focus to support internal investment decisions and to allow B2B customers to understand the longer-term perspective before they make their investment decisions. A product example is a new middleware software that is supposed to run on-premises, like Docker before its first release.

Cruise Ship SPM focuses here on extending the product scope and thereby increasing the target market. Since customers do not want to test and install new releases in the customer-controlled environment often, the frequency of releases is low, often one or two per year. As a consequence, the new and changed contents of these releases is more significant and requires thorough release planning based on analytical requirements engineering. Product strategy and roadmap continue to be important as is life cycle management. A product example is the Oracle database.

These four scenarios may not fit each and every situation. However, from our experience they are quite representative. We will refer to the four scenarios in this book whenever this differentiation is needed.

When software organizations adopt agile approaches, they use often Scrum as a basis that they adapt to their specific needs. For a detailed description of Scrum, we refer to the Scrum Guide [SuthSchw13], books like [Rubin12], and articles like [Kittlaus12]. Scrum was originally developed and described as a methodological framework for software development. There it has proven to be valuable in a lot of scenarios.

Scrum knows the role of a product owner, a member of the Scrum Team. The product owner feeds the team continuously with work that is specified with a prioritized list of user stories, called backlog. The user stories correspond to requirements. The product owner is the interface to the outside world and shields the rest of the team so that it can focus on development with optimal productivity. Implemented like this, the product owner is a rather operational full-time role whose tasks overlap with a software product manager's in the areas of requirements engineering and release planning.

This overlap needs to be sorted out in terms of process and role definitions. Some Scrum consultants claim that the most productive solution is one person who assumes both the product owner and product manager roles and all tasks attached to them—which they call product owner. Unfortunately, this is wishful thinking for most organizations! The approach may work in some environments with one Scrum Team, but it can hardly scale up. The poor person who gets this combined product owner and product manager role will always be pushed by the team to prioritize his operational tasks. Over time, the more strategic tasks are neglected—to the disadvantage of the product and the organization. Alternatively, operational tasks can be delegated to other Scrum Team members, but that boils down to an implicit split of the two roles again.

In most environments, it makes more sense to have the two roles product owner and product manager separated. They should remain strongly linked for optimal communication. They should have clearly defined distinct tasks and responsibilities (see [Kittlaus12]) that are comparable to those of the strategic and technical product managers. Product owners can either be part of the Development organization with a strong connection into SPM or be a part of the SPM organization that is delegated into the Scrum Teams (see [Leffing11]). For larger organizations that want to scale up Scrum, Leffingwell's Scaled Agile Framework (SAFe) is the most popular approach [Leffing16, KnasLeff17].

In the scenarios where the vendor controls the runtime environment, agile and lean approaches are dominant and increasingly extended into continuous delivery, which is based on continuous development, integration, and deployment (see [HumFar10]). While this approach can result in significantly improved time-to-market, it requires high investment in software architecture. Its success depends on fine-grained service-oriented architecture, on test automation, and on infrastructure such as the sophisticated configuration management and deployment system Apollo of Amazon (see [Vogels14]).

For corporate IT organizations, the four software product scenarios also apply, but instead of using the control of the runtime environment as one criterium, it makes more sense to look at the criticality of the software product for the organization (see Fig. 2.11).

Software Product Scenarios		Life Cycle Phase	
		New Product Revolution	Existing Product Evolution
Criticality	Non-Mission-Critical and Less Regulated	Powerboat	Speedboat
	Mission-Critical and/or Highly Regulated	Icebreaker	Cruiseship

Fig. 2.11 Software product scenarios for corporate IT organizations

For mission-critical and/or highly regulated areas Development and Operations are usually strictly separated. Rigorous processes have been established to bring new and changed software code into the runtime environment with a rather formal handover and a lot of checks and balances. By doing it this way, these organizations have managed to reach ambitious quality and security objectives even with quite monolithic software applications, typically back-end legacy applications. For non-mission-critical and less regulated areas, the focus is on tighter integration of Development and Operations. This approach is also known as DevOps, that is a prerequisite for continuous delivery approaches described above (see [BasWebZh15]). Companies can apply this, for example, to most front-end applications and newly developed non-critical applications. A lot of corporate IT organizations use hybrid approaches by running older applications as cruise ships. They start new non-critical application development and deployment as powerboats and later running them as speedboats (see the highly entertaining novel [KimBeSp14]). Mc Kinsey calls this a two-speed approach [AveBePe15].

2.6 The Role and Organization of SPM

A software product manager handles the management of software with the objective of achieving sustained success over the life cycle of the software product, family, or line. This objective refers to economic success, which is ultimately reflected in the profits that the products generates. Software product managers have the business responsibility across different versions, variants, and associated services of a product. They have to manage a broad set of product-related activities as shown in the ISPMA SPM Framework (Fig. 2.9).

2.6.1 Objectives and Success Measurements

Sustainability is a common corporate management objective for companies aiming at ensuring their long-term existence, even if investors have been demanding ever more short-term success. Sustainability should be included in the corporate vision statement and strategy (see Chap. 3) and become part of a lasting business model. For instance, an IT consulting firm cannot suddenly become a software vendor company and vice versa. Such a change even if feasible would take several years.

Successful products are not automatically sustainable. In the music business, for example, concert tickets are seasonal articles that lose any value when the concert is over. Companies who promote or arrange concerts can have a sustainable business model and a sustainable product family, but their individual products are not sustainable. Furthermore, a music group can serve as a brand name, so to speak, and thus be sustainable.

The examples shows that even if a company has an inherently sustainable business approach, its individual products will not necessarily be sustainable, but its product families and business model may be. Sustainability is found in a

company's assets, i.e. in the company's true values, that need to be protected and developed. Together, the assets will determine the company's sustainability. The purpose of product management is to manage such product-related assets systematically on a long-term and sustainable basis, regardless of whether this involves single products, product families or product platforms.

In the case of software products, the single product itself is usually sustainable, at least for a time. An exception are low-priced consumer products, such as computer games. Here, the individual product may enjoy a burst of popularity and then fade away. The game product family or platform is the determinant for sustainability and should be managed accordingly. Software product management refers to the management of a software product (or product family or platform) over its entire life cycle in accordance with corporate level objectives.

Corporate management may press strongly for short-term success. It may routinely assign higher priority to exigencies than to essentials. Such pressure generates the need for caretakers whose job it is to pursue and attend to all urgent major and minor demands in the company. In our opinion, such caretaker tasks are best organized in a staff function like a "management assistant." Such an assistant can deal with them across the organization. Alternatively, corporate management may be geared more to a sustainability concept. In that case, it will need people with managerial skills to manage the corporate assets according to the strategies and objectives that are laid out. It is the software product manager's task to manage these assets insofar as software products (or product families or platforms) are involved.

In reality, of course, a company is never quite as black-and-white as these described models. The emphasis on sustainability does not imply that a company need not quickly and actively take care of urgent customer problems, for instance. Nevertheless, we find such an exaggerated comparison useful for defining software product management objectives and job descriptions. A software product manager cannot work strategicly on software products as sustainable company assets, and at the same time be burdened with endless day-to-day tasks. This combinationwould only create another caretaker function with a new name. Such a function can be useful and important for the company, but it would not be software product management as defined in this book. Alternatively, the software product management function can be staffed with enough personnel so that employees can be assigned to individual tasks, with caretakers, branding, business analysts, and requirements management specialists all cooperating. In any case, software product managers, as defined here, and caretakers must work closely together.

In our view, the primary objective of software product management is to achieve sustainable success over the life cycle of the software product (or product family or platform). This objective refers to economic success, which is ultimately reflected by the profits that the product generates. Since profits lag behind investments, e.g. an investment phase involving losses may be followed by an extended highly profitable phase, customer satisfaction is often considered as a significant measure of software product management success. This significance is based on the hypothesis that there is a strong correlation between customer satisfaction and customer

loyalty to the product and the producer. According to recent publications and based on our experience, this is only partly true (see [Reichh96]). Of course, an extremely dissatisfied customer is likely to switch to a different product and producer. Conversely, a high degree of customer satisfaction will not guarantee that a customer will not switch to a different product or producer. However, only if the functionality of a replacement product is much better (e.g. it is considered to be a "technology leapfrog") or one is very dissatisfied with the current product, is it worth the risk of changing. So there is value in customer satisfaction, and it is an important factor to be taken into consideration. However, the measurement of the concept is challenging as we will outline in Sect. 5.6.

In certain situations and with some products, software product management does not focus primarily on profits attributable to a single product. For example, it could be in the interest of the company to achieve a maximum number of installations of a particular product platform as a prerequisite for selling other profitable products. One example of this has been the pre-installation of the Microsoft Windows operating system on new computers, for which the computer manufacturers pay Microsoft a relatively small license fee. By doing this, Microsoft ensures the continued dominance of its Windows operating system on desktop and laptop computers, which in turn serves as a platform for a large number of profitable software products sold by Microsoft and other vendors. In this case, market share is a better measure than profits. However, it needs to be taken into account that no one ever made much money from a high market share at a low or negative profit.

Two things become apparent in this discussion:

A conflict exists between software product management and executive management. Product management, by definition, has a long-term focus. Executive management has the desire to define objectives and variables that can be linked to business periods and thus evaluated annually. Development cycles or product success may not fit neatly into the business cycles that management will want to use for checking whether product management is on the right course and, if necessary, to make corrections. Also, there is the need to set individual objectives for each staff member and meet with them once a year to discuss performance and a salary increase. Unfortunately, this conflict is usually resolved in favor of more short-term objectives and measurements.

The sentence that a high-ranking American manager had hanging up as a poster in his office saying "Measurement Systems Do Work!" is true. If a company uses measurement systems to evaluate the performance of its employees and perhaps even makes bonuses and salary increases dependent on the results of such an evaluation, it must assume that the employees will try to optimize those measurements. This optimization means that, unless the measuring system uses measures that reflect what is intended exactly, the employees will be optimizing the wrong things. Since measurement systems are supposed to be simple so as to minimize the amount of time and effort necessary for evaluation, we often observe that the objectives become distorted and the wrong things are optimized. The inadequacies often have absurd consequences. Sometimes software licenses are unnecessarily given away to achieve certain target values with respect to the

number of licenses installed. This debases the value of the product in the marketplace and the price it can command. More work is sometimes invested in the manipulation of figures to meet poorly conceived measurements than in actual business activities.

We do not have any easy solution regarding the topics of "objectives and variable elements of compensation". In the end, the only possible solution is for software product managers and their superiors to talk about the problems we have described and reach an agreement together. This agreement should include important, period-linked objectives regarding actual individual employee tasks. Yet at the same time they should ensure that the product manager will still be able and willing to pursue the objective of sustainable product success. There are some available levers: milestone payments, employee rankings, longer term quotas and evaluations, career enhancement (many levels in a professional position). Richard Campione, former SAP's Senior Vice President of Suite Solution Management, confirms this: "The core concern is that the measurable key performance indicators (KPIs) tend to be significantly lagging indicators, and so while good for communicating intentions, and useful for long term, they frequently are inadequate for the short term. Here one needs to blend the solid, quantifiable KPIs with softer measurements and people's judgment." He uses criteria like product usage, deliverables, market responses to the product, and 360-degree evaluations involving responsible counterparts in the other units of the company like Sales, Marketing, Development, etc.

2.6.2 The Role of the Software Product Manager

If software product management is implemented as a role in the company organization, corporate management must define the purpose and objective of this role and communicate it throughout the company. It also must ensure that all those concerned find the job description acceptable. Regardless of how a company is organized, responsibilities will always be incompletely defined. Some issues and problems will always be neglected because no one feels responsible. Product management is therefore often misunderstood as being a universal caretaker responsible for all such issues and problems. Corporate management must deal with this problem by defining and delimiting the scope of all relevant tasks. It also must ensure that the software product management function is instituted with a genuine focus on sustainability.

The software product manager is supposed to be the person chiefly responsible for all relevant aspects concerning his product. Management skills are individual, personal qualities, but they can also be backed up organizationally. A management position tends to be easier to fulfill when it has managerial authority. Product managers frequently lack such authority, however.

The following chapters detail the broad spectrum of issues and tasks that a software product manager stands responsible for. Irrespective of the scope of his managerial authority, the software product manager has a cross-organizational role,

requiring a high degree of communication and coordination between all functional, organizational units. Condon describes this challenging task in detail [Condon02].

A job description should include the skills required for this position, i.e. the knowledge and experience needed to perform the tasks associated with the job. It will reflect the scope of responsibility of the software product manager's position as discussed in this chapter. This approach to the software product manager position is problematic because the scope of responsibility is so extensive that it will hardly be possible to find a candidate who can meet all the qualifications. Richard Campione, formerly SAP's Senior Vice President of Suite Solution Management, says: "If you try to find a person who fulfills the complete list of requirements you end up with the null set." This statement does not mean that a "software product manager" position should not be established and filled. Such a position should rather be created with the understanding that a candidate cannot be equally experienced and knowledgeable in all aspects of the job. It is a management position for which it is more important to work in cooperation with specialists from all relevant organizational units, ask the right questions and be able to draw conclusions. Basic managerial skills, extensive knowledge and experience in two of the relevant subject areas, and a capability of seeing things in a broader perspective will usually be an adequate basis for performing the job. Software vendors always find it difficult to fill product management positions, since there is no specific training for the job. This difficulty typically leads to a mixture of very experienced people in such positions who have worked in completely different areas of the company or industry during their career on the one hand, and specialists who can competently assume responsibility for a certain subtask due to their training and career, e.g. branding, on the other. This mixture is particularly visible if the software product management function is staffed with enough personnel to allow a specialization by skills. In any case, career entrants are seldom seen here.

Software organizations can have different foci for the role of SPM which translates into different priorities regarding these three areas:

- Business aspects.
- Product contents.
- Product Marketing.

We see three typical manifestations in the industry that are shown in Fig. 2.12. The size of each circle represents the priority of the area.

We consider Product Marketing Manager as a role separate from SPM. However, in some organizations both roles are assigned to the same person and may be called SPM. The strategic SPM is the business leader or "mini-CEO" [Ebert07]. The technical SPM is more focused on defining the contents of the product than the other two specializations.

Software Product Management's orchestration task—as shown in the SPM Framework (Fig. 2.9)—means to optimize the cooperation of all other units to achieve product-related goals. This task is conflict-laden in several dimensions. If the company has more than one product or product family, there is competition, not

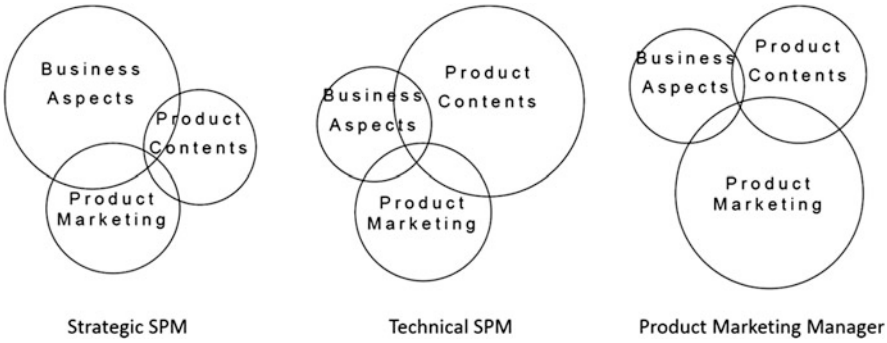

Fig. 2.12 SPM manifestations

in the market, but internally regarding the limited resources. In this competition that is carried out in strategy discussions and planning processes, each product manager represents his product. He is expected to take a strong position and be biased. On the other hand, it may not help his career if he questions the executive management's strategy. These inherent conflicts are the reason for why the product manager needs not only broad competence, but also a high degree of diplomatic dexterity when dealing with conflicts of goals and culture, human factors, and corporate politics. This aspect is well described with real world examples by Condon [Condon02]. Executive management can and must help here by clearly defining and separating tasks, responsibilities, and competencies (see [Gorche11]) and by acting as referee in case of escalation.

Corporations use various titles for the software product manager's position, such as product manager, program manager, solution manager, offering manager or brand manager. In corporate IT organizations, terms like "application manager" or "(application) service manager" can also be found. The latter comes from ITIL role of the same name [Axelos16]. Essentially, most of these names refer to the software product manager described in this book, although some differences do exist.

2.6.3 Organizational Aspects of SPM

We have already indicated the broad spectrum of topics and tasks that a software product manager has to handle. Software product management is not primarily focused on a single development project or a single marketing action, such as a product launch. These are "merely" steps taken in the pursuit of long-term sustainable objectives. Software product management is the combination of all the tasks described in this book. It can therefore be conceived neither as a project which by definition would have a beginning and an end nor as a process which by definition would consist of a well-defined sequence of process steps. Only some tasks can be interpreted or organized in this manner. Requirements management, for instance,

can be described as a process with respect to individual requirements (see Sect. 4.2), whereas the development of a new product release is often considered to be a project.

In conformance with the sustainability of the software product (or product family or platform), software product management viewed collectively as a combination of tasks is an ongoing activity. This fact is rather unpopular at a time when a lot of people believe that they can measure the efficiency of an organization by the share of its project or agile work. This idea stems from a vague feeling that unproductive colleagues might just comfortably while away their time performing ongoing tasks, whereas strictly organized project work—maybe based on agile methodology—would force everyone to be productive. In over 30 years of experience in the software business, we have not found any reason to support such views. We have experienced many unproductive projects and highly productive employees who perform ongoing jobs. Furthermore, ignorance of ongoing critical tasks, such as market analysis or personnel development, tends to create enormous problems in the medium term for the company as a whole. Our observations show that good corporate management and a good working atmosphere help significantly to influence productivity. Software product management, in particular, is a good example of an ongoing task that is one of the most challenging jobs in a company. It requires a high degree of personal commitment and diligence for the product manager to be successful in the medium and long term.

A number of product management tasks, in particular in product strategy, are tied to the company's annual cycle for strategy and financial planning. For these tasks, an annual schedule may help product managers to prepare their inputs in time for the company's planning schedule. However, there are other tasks like dealing with legal aspects that are often triggered by outside events like a customer situation. And some tasks like performance management are really continuous tasks, i.e. require the product manager's attention at least once a month.

In big companies, product managers can be organized in a team that can exploit the different skills of the team members with an intelligent division of work. In a small company, such a division of work in a product management team is usually not possible. In addition, the success of a small company is so tightly connected to the success of the one or the few products that executive management handles the most important product management tasks themselves, particularly the financial ones. Still, the holistic product view is important for smaller companies as well. An single executive cannot cover the holistic approach for time and skill reasons and might neglect core elements like product planning and legal considerations.

The holistic entrepreneurial component in software product management suggests a company should be structured so that the SPM unit is close to executive management, i.e. as a staff function. The probability of success of this structure depends on the company's culture and the attitude of the people involved, in particular of executive management. The staff function has a lot of responsibility without the right to give orders to those who have the resources and need to contribute for success. If the company's culture is in conflict with non-hierarchical cooperation and any escalation to executive management is

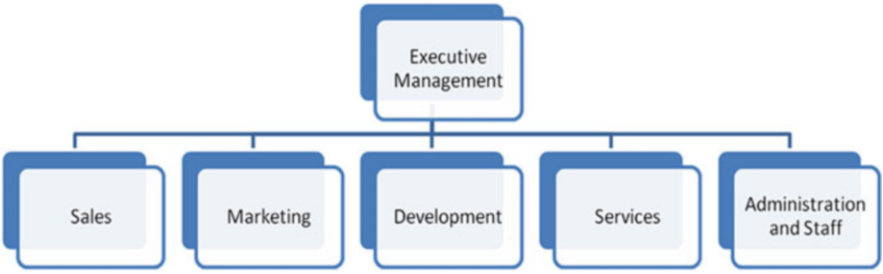

Fig. 2.13 Functional organization

interpreted as a personal attack, as is common in a lot of companies, the software product manager is in a losing position. SPM as a staff function can only work if executive management defines, communicates, and supports the software product management unit openly.

What are the alternatives? Let us first assume a functionally oriented organizational structure (see Fig. 2.13). Here, underneath the corporate executive management there are the areas of Development, Sales, Marketing, and Services plus administration and staff functions, e.g. Finance and Human Resources. The software product management unit could either be integrated with Development or Marketing. An integration in Sales is not recommended since the short term goals of Sales create too much of a conflict with the Software Product Management's goal of sustainability.

Integration into Development often seems more plausible. This approach is driven by the idea that Development can best foresee where technology is going. The experience of IBM that used this organizational structure for a long time shows disadvantages, however. Developers are typically far away from the market and the customers even when diverse initiatives support customer contact. Cultural difference between Development, Sales, and Marketing prevent a product management unit that belongs to Development to fulfil its role as mediator. It is not considered as (and probably isn't) impartial. The responsible Development manager will typically avoid being escalated by his Product Management unit. And Sales and Marketing—having been cut out of the requirements and implementation decisions—may well disavow the product after it is developed.

The same disadvantages appear in part when one considers making Software Product Management a part of Marketing. Marketing has the advantage of being closer to the market and the customers. Marketing is usually a smaller unit whose self-interest is better aligned with sustainable software product management, although its goals are more short to medium term. So we argue for putting the Software Product Management unit into Marketing if executive management does not want it as a staff function in a functionally oriented organizational structure. However, this is only valid if Marketing is clearly separated from Sales.

A lot of companies have migrated from a strictly functionally oriented structure to other organizational structures that are better aligned with the necessities of

cooperation across the company. It is not the subject of this book to discuss the pros and cons of these organizational structures that range from matrix to business process orientation. However, we want to emphasize that, given the diverse forms of cooperation and communication within a company, no organizational structure—whatever it looks like—can make non-hierarchical, horizontal cooperation superfluous. Exactly the management of such horizontal cooperation is a major task of product management called orchestration.

Product Strategy

3

The pace of technological change in the past several decades has been faster in the IT industry, specifically in the software sector, than in any other industry. This development makes it necessary for the organizations concerned, both software vendors and corporate IT organizations, regularly to make far-reaching decisions under uncertainty that have considerable financial and even survival consequences. Yet, in spite of the fast pace of change, companies with a clear strategic view are the ones that prove to be successful in the long term. In recent years, SAP, Apple, or PTC have been excellent examples. This does not mean that at these companies all product ideas are succesful or that every product strategy produces the desired results. It does also not mean that strategy definition is strictly top-down without experimentation. However, it does mean that these companies routinely manage to reach agreement on and consistency in their corporate vision, corporate strategy, product strategies (or product platform and family strategies) and more short-term implementation plans by means of iterative processes that sometimes require a great deal of time and effort. So the companies realize faster what works, and what does not work, can avoid waste, and move on in a more focused and more aligned way.

While a product vision gives direction for the future of the product in a condensed way (see Sect. 3.1), the product strategy provides the details by which to implement that vision. Normally a strategy covers a time span of about 1–5 years in the future. For consumer (B2C) products, it is typically shorter, for business-to-business (B2B) products longer. Due to the increasing pace of change, a lot of companies have shortened the strategic timeframe that they look at, because specific strategic plans for the outer years have often proven to be obsolete by the time you get there. We consider this as a dangerous trend. In our view, companies as well as product managers need to know on a more abstract level where they want to go. Otherwise long-term decisions are made without any foundation which increases risks further. So having more abstract strategy contents for the outer years is better than no contents at all. In the sections of this chapter, we shall come back to this challenge. In the Powerboat and Icebreaker scenarios (see Sect.

H.-B. Kittlaus, S.A. Fricker, *Software Product Management*,
DOI 10.1007/978-3-642-55140-6_3

2.5)—when a new product is developed—the strategy usually evolves in parallel to the first version of the product with experimentation and learning.

The software product manager is responsible for defining the strategy for his product (or platform or family) and to support and update this strategy using a standard process over time. The strategy should address the development or evolution of each of the items in the list shown in the Product Strategy column of ISPMA's SPM Framework (see Fig. 2.9) during the strategic timeframe:

- Positioning and Product Definition: define scope, customer value, target market segments, channels of the product.
- Delivery model and service strategy: explain how the software product is made available to customers, and which professional services will be offered as part of the whole product offering.
- Sourcing: define where resources, in particular human resources and software components, are coming from.
- Business case and costing: define business elements like benefits and cost, forecasts and the business plan.
- Pricing: determine how prices are defined and managed over time.
- Ecosystem management: define the roles to be played in relevant ecosystems, and managing relationships with partners and other external stakeholders.
- Legal and IPR management: take care of all legal product aspects.
- Performance and risk management: define and take actions based on continuous business performance measurement and risk assessment.

These items are, of course, interdependent. If, for example, the available budget is smaller than originally assumed, it will only be possible to expand the product scope to a lesser extent or more slowly. If new segments are to be added to the target market within the strategic time frame, the product scope may have to be expanded.

Dependency on other products can also have considerable consequences, e.g., if certain functionalities or enabling code must be available in several products at the same time. In addition, the aggregate resource planning for all products needs to be coordinated with the resource planning for the company as a whole. As a rule, the larger a company is and the more dependencies of this type exist within a company, the more difficult, complex and time-consuming the entire alignment process will be, in particular for budgets, resources and roadmaps. The roadmap can be considered as the bridge between strategy and implementation plan. It will be covered in Sect. 4.1.

There are a number of models that can be used when defining parts of a software product strategy. One is Michael Porter's Five Forces [Porter79]. Another one is Alexander Osterwalder's Business Model Canvas ([OstPign10], see Fig. 2.2). Further variants of these models have been developed and are in use.

In Fig. 3.1, the elements of the Product Strategy column in ISPMA's SPM Framework (Fig. 2.9) are mapped to the Business Model Canvas and to the sections of this chapter which have a finer granularity in the areas of positioning and product definition.

Section in this book	Business Model Canvas	ISPMA SPM Framework, Product Strategy column
3.1 Product vision	./.	Product Definition
3.2 Product name	./.	Positioning and Product Definition
3.3 Customers	Customer Relationships Value Propositions	Positioning and Product Definition
3.4 Market	Customer Segments	Positioning and Product Definition
3.5 Product Definition	./.	Positioning and Product Definition, Delivery Model and Service Strategy
3.6 Positioning	Channels	Positioning and Product Definition
3.7 Service Strategy	Key Partners	Delivery Model and Service Strategy
3.8 Sourcing	Key Resources	Sourcing
3.9 Business View	Cost structure Revenue streams	Business case and costing
3.10 Pricing	Revenue Streams	Pricing
3.11 Ecosystem Management	Key partners	Ecosystem Management
3.12 Legal aspects	./.	Legal and IPR Management
3.13 Performance and Risk Management	./.	Performance and Risk Management

Fig. 3.1 Mapping of syllabus chapters, Business Model Canvas and ISPMA SPM Framework, Product Strategy column

In the last Sect. 3.14, organizational aspects, tools and options for documentation in the product strategy domain are discussed.

Software is becoming a core element of the products of more and more industries, like automobile, manufacturing, or health. Therefore, software plays an increasingly important role in the product strategies of companies in these industries. To some degree, for them product strategy becomes software product strategy.

3.1 Product Vision

3.1.1 Overview

A strong product vision is often essential to engage and convince all stakeholders inside and outside of the company of the worth of a product. A vision describes what the future product will be, why it is needed, and why it will be successful (see [McGrat01], p. 1 ff). The elements of the product strategy provide the details that

turn the vision into a manageable and executable path into the future. In bigger companies a product vision needs to be aligned with the corporate vision.

The first version of the product vision is needed when work on the development of the product's first version starts. Over time, the product vision continues to evolve so that it always looks ahead at least to the end of the strategic timeframe.

This section describes the contents of a product vision and outlines approaches how to get to a good product vision. The reader will understand the related process and the impacts.

3.1.2 Concept

Like product name (see Sect. 3.2), a product vision can be considered as an element that is referenced in a product strategy, but comes before it. The vision represents the objective that shall be achieved with the ideas set forth in the strategy. There is no universally agreed definition of product vision to be found in the literature. We define

Product vision = Conceptual description of what the future product will be at the end of the strategic timeframe, i.e. high-level descriptions of a product concept and a corresponding business model.

The product vision is the "guiding star" for the strategy. It outlines in a condensed form:

- Conceptual image of what the future product will be,
- The customer value proposition that says why the product is needed and cannot be replaced by an alternative, and
- The business value for the vendor, i.e. why it will be successful.

A product vision is focused on a point in the future, at least the end of the strategic timeframe. It is documented in a few sentences or as a relatively short text, no more than one page of paper. The vision needs to be phrased from a marketing perspective, in a style that has a motivating effect on stakeholders inside and outside of the company by painting a desirable, ambitious, but achievable future.

The product vision is especially important in the start phase when the first version of the product is conceived and developed. In case the product has not yet been named at that time, most companies work with an internal code name that is used in the product vision and in internal communication (see Sect. 3.2).

The term "product vision" is also used in connection with agile methodologies, in particular Scrum. While it is not mentioned in the Scrum Guide [SuthSchw13], most Scrum consultants consider it as an important element of the Scrum approach. Originally that agile product vision was focused on the given development task. It was supposed to describe in a condensed way what the Scrum Team was asked to develop within a couple of weeks or months. Over time, some Scrum consultants, e.g. Pichler [Pichler10], have broadened the scope and use product vision in the sense we define it here.

3.1.3 Development of a Product Vision

A convincing product vision looks very straight-forward, logical and easy. However, developing it is more difficult and time consuming than most people expect. We suggest the use of a template that we have used in a number of software companies. It can support the development of a vision by focusing on the problem space and the solution space. The problem is described in a solution-neutral manner and explains the pain-points addressed by the solution as well as the criteria used for evaluating product success from the customer perspective. The solution is described in terms of use scenarios, features, benchmark, and unique value proposition. For the development of the vision, a combination of the product manager's draft as a synthesis of ideas and contributions solicited internally [GoFrPaKu10] and a workshop approach [Chesbrou05].

Figure 3.2 illustrates the template. The example is drawn from a product that tracks consumables in an operating theatre. This template makes sure that relevant information is collected, but does not automatically result in a wording that is marketing-oriented. So in a second step, this needs to be turned into a marketing statement. This transformation can be made by re-ordering the presentation of the vision statement:

> In order to significantly decrease clinics' effort and increase the availability of operating theatres, the **Consumables Tracking Solution (CTS)** reduces the nurses' and analysts' manual work by tracking the use of consumables in an operation, enabling its analysis, and automating reporting. It enables clinics to increase the efficiency of the operation work and deliver decision-support for consumable planning and improvement.

Often companies want shorter vision statements like:

> The **Consumables Tracking Solution (CTS)** increases clinics' efficiency and reduces cost by automating tracking, analysis and reporting of consumables in operating theatres.

Problem statement	
the problem of	immense effort for reporting consumables
affects	nurses, and the clinic overall
the impact of which is	inefficient use of operating theatres
a successful solution	increase the availability of the operating theatres.
Position statement	
for	nurses and analysts
who	administrate, assist in, and improve operations
the	**Consumables Tracking Solution (CTS)**
that	tracks the use of consumables in an operation, enables its analysis, and automates reporting
unlike	the current labor-intensive manual approach
our product	increases the efficiency of operations and delivers decision-support for consumable planning and improvement.

Fig. 3.2 Example of a problem and position statement (source: EU FP7 FI-STAR)

In order to achieve the buy-in of the product team members, it can help to develop and evolve the product vision in workshops with the key team members.

3.1.4 Further Examples and Variations

While it should be fairly easy to come up with a convincing vision statement for a new product, it can be challenging to do that for a product in a later phase of the life cycle when investments are typically reduced. The following examples illustrate this dependency.

CRM SaaS product, early life cycle phase:

> For a mid-sized company's marketing and sales departments who need basic CRM functionality, the CRM-Innovator is a Web-based service that provides sales tracking, lead generation, and sales representative support features that improve customer relationships at critical touch points. Unlike other services or package software products, our product provides very capable services at a moderate cost.

Code Generator product, late life cycle phase:

> "Current drivers of the Code Generator vision include:
> - Continued support of J2EE and .NET frameworks
> - Extending support of Web services
> - Infrastructure enhancements
> - Integration with our companies' systems management, security and application life cycle solutions

> The Code Generator is a world-class enterprise application development environment that continues to deliver the core capabilities it has provided for two decades:
> - Platform independence
> - Application portability
> - Productivity."

Some companies prefer not to stay so close to the product functionality and to be more "visionary" which translates into more abstract vision statements. Here are examples:

Salesforce.com started with "The end of software" as a vision which sounds a bit pathetic and is misleading since SaaS is based on software as well. What they have meant is "the end of deployed software". They used this claim to justify their software-as-a-service approach.

SAS Institute's "Providing organizations with the Power to Know" fits the business intelligence market nicely promising the transformation of data into insight.

3.1.5 Outcome and Impacts

A compelling product vision can be a powerful instrument to keep the product team aligned and on track, especially during the development of the initial version of the software product. It can also be a good marketing tool during initial product launch and during later phases of the life cycle that communicates the core direction for the product.

A number of studies have documented the important role of product vision. Pearce and Ensley [PearEnsl04] looked at teams working on innovative development, shared vision and innovation effectiveness which they define as the speed at which innovation is developed, the magnitude of the innovation, and the productivity of the implementaton. They found shared vision and innovation effectiveness to be reciprocally, positively and longitudinally related. This means they influence each other positively over time. Tessarolo [Tessarol07] looked at cross-functional teams working across internal and external organizational boundaries. He could show a shared clear product vision to be strongly correlated to speed of development.

Lynn e.a. [LyAbVaWr99] studied new product development projects in high tech industries. They showed product success to be most strongly correlated with vision and new product development process. In [LynnAkg01] Lynn and Akgün further detailed these findings by looking at four scenarios and three vision factors, vision clarity, vision stability, and vision support, i.e. how strongly the project team members shared the vision. They found that vision clarity correlates with success in evolutionary market and technical innovation and in revolutionary innovation, but not in incremental innovation. Vision stability correlates with success in incremental innovation and evolutionary market innovation. And vision support correlates with success in incremental innovation and evolutionary technical innovation.

3.1.6 Summary and Conclusions

Given the strong impact a good product vision has on product success and speed of implementation, it is worth investing some time and effort into creating the vision and revisiting it whenever the product strategy is updated. The proposed template can help with articulating a convincing vision. The product strategy must be tightly linked to and synchronized with the vision. So an update of the vision needs to be reflected in the product strategy.

3.2 Product Name

3.2.1 Overview

The product name is both the internal and external "face" of your product. So it should be chosen with a strong marketing perspective taking into account how the name as a term will be perceived literally and psychologically by all the stakeholders. With software products, often numbers are added to the name to differentiate versions and releases of the product. Such an approach both preserves the product branding and conveys newness with a higher number. Brand recognition may be significantly increased by combining the product name with a specifically designed logo.

3.2.2 Concept

A product name is used in a product strategy, but is not part of it. If in the start phase when the first version of the product is conceived and developed, the product has not yet been named, most companies work with an internal code name that is used in internal documents and communication. It is in the DNA of any language that people need a name as an identifier for "a thing" when they want to talk about it. There are lots of studies that prove that a good product name has an effect on the buy decision of customers, more so in B2C, but to some extent also in B2B, e.g. [HiAlCeBa13]. When the first version of the product is launched, the name, maybe accompanied by a logo as a graphical representation, must have been chosen and becomes the center of all marketing messages.

Naming a product works the same way for software as it works for other products. There are a number of criteria to be met for a good product name:

- Memorable, appealing and motivating for the potential customers in the target market.
- Distinguishing against competitive products.
- Legally on the safe side.

Naming is a major part of the branding of a product. Since software product managers are usually not experts on branding, it is advisable to involve such experts, in particular on the legal side (see Sect. 3.12).

The product name is what stakeholders identify a product by. So as long as the name does not change, they consider it as the same product even though it may change significantly over time. For licensed software products, the legal and financial views often differ from this public view, i.e. a new version is internally formally considered as a new product even though the name stays the same.

For licensed products, it is common to denominate software versions (see Sect. 4.3) after a specific nomenclature, which is generally dependent on the vendor,

however. Often, we find a three- or four-level hierarchy of software levels, for example:

- Version number.
- Release number.
- Modification Level.

The version identifier is added to the product name. An example of complete product identification is IBM z/OS Version 1 Release 9 Modification Level 5 (in short z/OS 1.9.5).

3.2.3 Process

Naming a product may sometimes be quite difficult. Everybody feels competent, everybody has an opinion. As the responsible product manager, you can leave the question to your executive manager who will dominate the discussion anyway, or you can organize a brainstorming or an internal competition with an award for the winner. A clearly articulated value proposition can inspire the name finding. The criteria to determine the winner are the ones listed above. A helpful discussion on project naming that is largely applicable to product naming can be found in ([GausWein89], p. 128 ff.).

Finding an internal code name is not restricted by legal considerations. For the official external product name, however, the legal criterion can easily turn all the effort that goes into brainstorming to waste. You do not want to spend a large amount of marketing money on establishing a new product name in the market, and then find out that somebody else has a trademark on it and forces you to change your product name. So legal clearance and possibly protection is highly advisable before the money is spent. Often the investment in a trademark for the geographic target markets can make sense. Relevant internet domain names need to be secured as well. Since most descriptive names are legally protected somewhere by a third party, you can either buy the rights to the name from that third party, or go with some artificial word as name that may not be descriptive but is not already legally protected. If internal brainstorming does not succeed specialized agencies can help who come up with proposals and take care of the legal clearance.

Once the name is set, the marketing activities are focused on making it known and creating a brand image in the target market (see Sect. 6.3). This usually means a significant investment.

Due to the significant branding investment, a change of the product name at a later point in time only makes sense in exceptional situations, e.g. when you are legally forced to change because some other party has the rights, or when the reputation of the product is so bad that you want to relaunch it under a different name, or when product names are aligned after an acquisition.

3.2.4 Summary and Conclusions

The product name is what people identify the product with. A good name can also have a positive effect on the motivation of people working on the product internally. While name finding is a hopefully rather infrequent task for a particular product, it requires and justifies some effort and investment. Consideration of legal risks is critical.

3.3 Customers

Customers are at the core of any business. No customer, no business. So anybody who cares about the business must care about the customers. We differentiate between customers as the ones who are the legal contractors and pay, and users who work with the software product on a frequent basis. In the B2B area, customers are usually companies that are represented by some select employees who are decision makers and/or interface with the software vendor, and are separate from users. In the B2C area, customers and users are often the same persons.

While the primary interfaces to existing and potential future customers are usually with Sales and Marketing units, the software product manager needs to be in touch with customers and users in order to better understand needs, usage of the product, and user experience with regard to the product. Every product manager has to find his own way of having frequent customer contacts without neglecting the wide spectrum of other tasks that are part of the product manager role.

In the B2B software business, it can make sense to maintain a couple of personal relationships to customer representatives that the product manager can informally approach whenever he wants feedback on specific questions. However, these relationships must not fully replace the broader consideration of the target market.

In the B2C software business, there is usually a mass market with a high number of customers. Feedback from individual customers can be even more misleading than in the B2B market. A representative sample of customers promises more insight. Nevertheless, one-on-one contacts at fairs or similar events can add value.

Customers of software standard products pursue conflicting objectives. They usually want the standard product to be a perfect fit for their needs and their environment, i.e. like a custom-made solution, but at the much lower price of the standard product. Finding a compromise between these conflicting objectives is a key success factor of the work of the product manager that touches on issues like customer value definition, product definition, customizing options, service strategy, pricing, or handling of customer-specific requirements. Conflicts can also arise between requirements benefitting existing customers vs. potential new customers, e.g. in different market segments.

The relationship with a customer can be considered as a “Customer Life Cycle”, which consists of four phases that have some similarities to the product life cycle (see Sect. 4.4). Note, however, that the former life cycle describes how a customer

Phase of the customer life cycle	Appropriate activities
Customer initiation	requirements management information gathering consulting configuration
Customer acquisition	negotiation transaction processing financing delivery
Customer loyalty	customer service (Service Level Agreements) user behavior warranty, maintenance updates cross-selling
Customer recovery	new business resale secondary purchase upgrade/up-selling

Fig. 3.3 Phases of the customer life cycle (based on [MehlBiSt14], p. 64)

evolves. The latter describes how the product evolves. Figure 3.3 shows a simple customer life cycle.

In each customer life cycle phase the vendor, i.e. primarily the Sales and Marketing people, ought to define appropriate actions in order to optimize, maintain, and renew customer relationships.

It is in the interest of the software vendor, and therefore the product managers of the involved software products, to make the relationship with a customer, i.e. the "Customer Life Cycle", last as long as possible. We recommend to establish measurements for success for the different phases.

3.4 Market

3.4.1 Overview

The concept of a market is a central element of our worldwide economic system, and also a central term in economic theory. In general, it is the meeting point of buyers and sellers of goods and services where they can exchange offerings against some form of compensation. This exchange is called a transaction.

For a particular product, it makes sense to focus on the subset of buyers and sellers that are relevant for the product. This relevant market can be further broken down into market segments that differ with respect to certain criteria, e.g. B2B versus B2C markets.

The sellers in a relevant market can be competitors offering similar or functionally overlapping products, or players with offerings that are complementary. The latter players are candidates for partnership.

For a software product manager, defining the relevant market is a key part of the task of product positioning usually done in cooperation with Marketing. Once the relevant market is defined, the buyers in that market are the target customers for the product.

3.4.2 Concept

Since "market" is such a central concept in both the academic and the real world, there are lots of definitions from micro- and macro-economics, marketing, sociology, or anthropology. For this book, we use these definitions:

Market =
(a) The area of economic activity in which buyers and sellers of goods and services come together, and the forces of supply and demand affect prices.
(b) A geographic area of demand for commodities or services.
(c) A specified category of potential buyers.

Market segment (of a given market) = Market with a subset of the buyers, sellers, goods and services of the given market.

The definition (a) above means that a market is a set of sets of potential buyers, potential sellers, goods and services, respectively, at a particular point in time. A market segment is homogeneous with regard to certain criteria.

As an example, we look at the market for weather apps worldwide. By using different criteria for segmentation, we can create different market segments of this weather app market:

- Free vs. paid apps: defines market segments that are restricted to the sellers that provide free or paid apps; with free apps, compensation is not monetary, but e.g. attention to advertisements. Some apps may be available with and without payment and therefore are part of both segments.
- Geography: restricts the market segment to the buyers and sellers active in a chosen geography and the products available in that geography, e.g. the UK.
- Functional scope: restricts the market segment to apps that cover the chosen functional scope, e.g. 15-day forecast in weather apps.

Sellers in a given market for software product A can be

- Competitors: they offer similar or functionally overlapping products; the buyer will either buy product A or the competitor's product, but not both,
- Potential partners: they offer products and/or services that are complementary to product A; they are members of product A's software ecosystem,

- Unrelated sellers: they are neither competitors nor potential partners; if there is a higher number of unrelated sellers in the market defined it is an indication that the market definition may not be sufficiently precise and narrow.

The group of buyers in a given market segment is often called customer segment, e.g. in the Business Model Canvas (see Sect. 2.4). Product managers need to be very specific about the customer segment(s) they are targeting so that they can develop a thorough understanding of customer needs. This understanding in turn is key to developing compelling value propositions that help sell the product (see Sect. 3.6).

A competitive advantage is achieved when the company offers a product that the competition does not, or when the company offers a better product than the competition. Competitive advantage comes from one of two sources:

1. Having the lowest cost in the industry or
2. Possessing a product/offering that is perceived as unique in the industry.

Another contributing factor is the scope of product-market (broad or narrow). A combination of these factors provides the basis for the following three types of competitive strategies:

- Low cost strategy (be the cost leader),
- Differentiation strategy (be unique),
- Focus strategy (be the niche leader).

Regarding competitive strategy, there is a rather static view of an industry's attractiveness, based on Porter's Five Forces [Porter08]:

1. Rivalry among existing competitors,
2. Threat of new entrants,
3. Bargaining power of suppliers,
4. Bargaining power of customers,
5. Threat of substitute products/offerings.

3.4.3 Determining a Product's Market

Determining the relevant market for a software product is by no means trivial. The most comprehensive definition of the market is the overall software market, which the software product manager can certainly use as a basis. However, as a whole, it is so huge and inhomogeneous that it is not really useful. The smallest conceivable market consists only of one product. This definition is not absurd if this product creates a new market without any competitors. In between these two extremes, of course, there are many ways to define the market that product managers play with in practice. If you want to point out the enormous sales potential of a planned product, for example, you select a broadly defined market segment with a correspondingly

high volume. If you want to demonstrate what a large market share your product has, you choose a narrower definition of a market segment. There are no fixed rules for deciding which market definition is the best for a product. Market research companies that analyze software markets often divide the overall market into multi-layered market segments by functions, geographic regions and customer groups (see Sect. 5.5).

There is a close connection between the market definition and the definition of the product scope which is part of product definition (see Sect. 3.5). The product manager must deal with this connection iteratively. On the one hand, the analysis of the market, market development and competition will influence the definition of the functional product scope. On the other hand, the product scope will determine in which segment of the market the product can successfully compete. What is needed, ultimately, is a time-related close correspondence between product scope, target market and business prospects. Planning the further development of such a correspondence over time requires coordination with the product vision, maybe also the corporate vision statement and strategy, and constitutes an essential aspect of product strategy.

The scope of requirements ensuing from the definition of the target market should not be underestimated. If the target market is the consumer market, the product must meet different requirements with respect to usability, packaging, pricing, sales channels, support structure, etc. than a B2B product. Some market segments are so special that they even require different approaches to development and requirements management. For games for example, development is often based on a story line rather than a technical specification, and developers typically have more freedom in the graphical design as part of an iterative prototyping approach (see [Waldo08]).

The requirements for a product to be marketed internationally are significant, e.g. in terms of product requirements, legal requirements and in the marketing and support structure. The resulting expenses will be offset by higher sales expectations. This relationship is discussed in detail by McGrath ([McGrat01], pp. 235–255).

Part of forming a successful product strategy requires identifying competitive advantage. Providing an answer to the question "How can the product satisfy the needs of potential customers better than competition?" needs to be incorporated in the value proposition. The following two methods can be used alone or together to identify competition:

- Industry method: This method of identifying the competition is based upon an already established industry in which the business operates. The competition is identified as companies that produce the same or similar products. For example, a car manufacturer would identify other car manufacturers as competitors, and a mobile service provider would identify other mobile service providers as competitors. This method is particularly important to identify competitors if the company is planning to enter an existing market. The industry method also takes into account the level of competition within the industry. Some of the

questions that can be asked to identify and characterize competitors in a given industry using this method are:
   a. Who else differentiates like my company?
   b. Who has entry and/or exit barriers like my company?
   c. Who has vertical integration like my company?
   d. Who is as global or as local as my company?
   e. Who has cost structures like my company?
- Market method: The market method depends largely on how well the customer need is defined in the description of the target customer segments. The market method of identifying the competition can be established upon marketing products or services to customer needs. The competition is identified as companies that satisfy the same need. Market analysis is supposed to identify market forces, industry forces and key trends (see Sect. 5.5) which can be analysed to identify competition. With the market method, a movie theatre may identify entertainment as the customer's need. Substitutes like video rental stores, amusement parks, and concert venues could then be identified as competitors. The main question to ask in this method is
   a. Who else can satisfy the same customer need?
   b. Who goes after the same market segments with comparable value propositions?

The Market method can also be useful if a company wants to re-segment existing markets or create a new market. In that case the product offering may not belong to any existing industry (i.e. there is no direct "peer-group" of products) for comparison. Petal diagrams instead of traditional X/Y axis competitive analysis diagrams can be used in this case (see Fig. 3.4) as suggested by Blank [Blank13b].

In the petal diagram, the company is placed in the center. Next, the identified new market segments from where the customers will originate (can be taken from "Customer Segments") are plotted as petals. After that each petal (representing a market segment) is filled in with the names of the companies that are representative players in that market. Finally, the current and projected market sizes of the adjacent markets can also be shown to analyze and discuss "How big will our new market be?".

It is important to consider

- Direct competitors who produce an almost identical product that is offered for sale within the same market,
- Indirect competitors who don't necessarily sell the same products, but offer different alternatives to satisfy the same customer need,
- Other alternatives customers have, including do-nothing or do-it-yourself.

Once the alternatives have been identified, gap analysis (surveys with the customers, distributors and partners) can be employed to analyze competitors, e.g. the strategic groups method. Strategic groups are segments of an industry that group companies with similar business models or business strategies

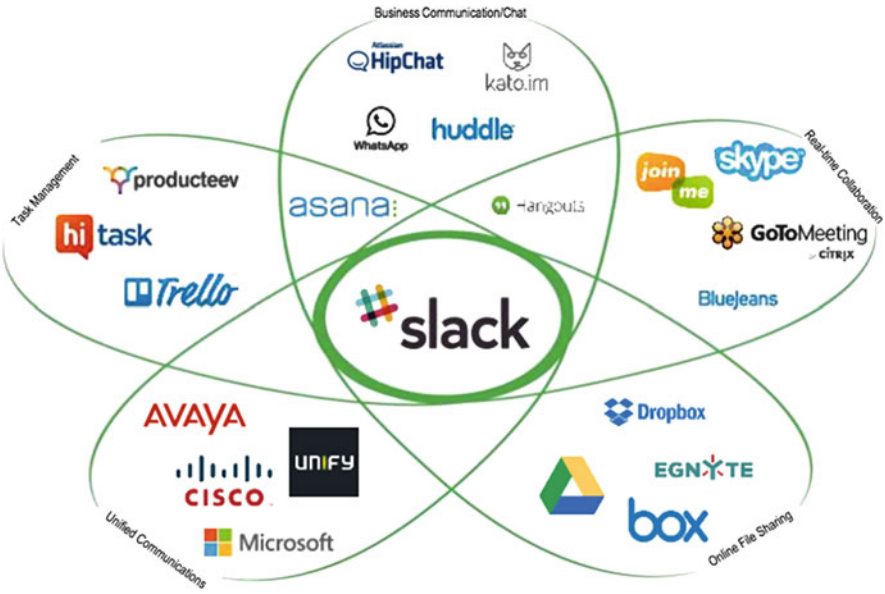

Fig. 3.4 Example for petal diagram, here for Slack (from http://cunninghamcollective.com/insights/blog/2015/06/30/escape-the-box-three-great-ways-to-use-the-petal-diagram-for-strategy)

[Porter98]. However, if the companies want to dramatically improve their value proposition, they need to identify offerings (not competing companies) that fulfill the same need (maybe in a different way).

A product manager usually needs inputs from other roles (senior strategy managers, marketing managers) to analyze competition and define competitive strategy for the respective product. A competitive strategy is defined as a long-term action plan that is devised to help a company gain a competitive advantage over its rivals. It consists of business approaches to attract customers (by fulfilling their needs), withstand competitive pressure and strengthen market position. A competitive strategy exploits competitive advantage by identifying ways to use resources and capabilities to differentiate the product from its competitors [FleBen02].

But for fast-changing software markets, more recent strategy models that emphasize the ability of companies to actively shape market boundaries and thus, create new markets (Blue Ocean Strategy, [KimMaub15]) may be more useful:

- Create uncontested market space,
- Make the competition irrelevant,
- Create and capture new demand (innovations),
- Break the cost/value trade-off,
- Align the whole system of a company's activities in pursuit of low cost and differentiation.

However, the two approaches are not mutually exclusive and can be combined. For example, by slowing down profit erosion with an effective competitive strategy for an existing market, a company can increase the funds available for Blue Ocean investments and consequently increase its chances of finding an untapped market with plenty of customers.

3.4.4 Variations

Customer segments do not always have to be very narrowly focused—according to Moore [Moore14, Moore04], it depends on the maturity stage of the respective market how broad or narrow the target segment can be. For example, when bringing to market a completely new type of B2B product—what Moore calls a new product category—it is often useful to kick-start mainstream adoption by focusing on a small, well-defined market niche (the beachhead segment). The strategy is to expand into adjacent market niches later—one after the other (bowling alley strategy). Once the new product category is better understood in the market, and broad adoption sets in at a fast pace (tornado phase), the vendor's top priority needs to be capturing market share. In this stage, a broader customer segment definition is preferable.

When using the Business Model Canvas as a tool for analysis, competition is a "blind spot", but an important topic for product strategy. Some variations of the business model canvas address competition more explicitly. An aspect sometimes important to consider is the number of customer segments covered: in some types of business models, for example in multi-sided markets (such as eBay or a real estate market place) it is obvious that at least two customer segments need to be covered—for example buyers and sellers in a market place. Since these customer segments are highly interdependent, they should be covered in the same business model canvas. In other cases, the product is simply targeting multiple customer groups, potentially with different value propositions. It can make sense to cover these multiple segments in the same canvas, too, especially if—apart from the value propositions—the rest of the business model is the same for all target segments.

Another important consideration is the relationship between products and solutions (see Sect. 2.2): in an organization that offers both individual software products and solutions as products that include these individual products, separate Business Model Canvases should be developed for each individual product and for each solution. In that situation, it may happen that an individual product is sold stand-alone and also gets included into multiple solutions. In that case, the question arises how to address the multiple solution relationships in the Business Model Canvas for the individual product (not on the solution level). If the different solutions target different customer segments, it might make sense to list these solution-level customer segments separately on the product-level canvas, with a separate value proposition for each segment.

Finally, many software products need to get buy-in from multiple different constituencies, for example a smartphone app for kids, where parents might make

the actual purchase or can veto the kid's purchase. In that case, it is often useful to include both the kids and the parents as separate target customer segments in the same canvas and to develop separate value propositions for these two segments. Other well-known examples for this are different roles that participate in the complex sales processes for large-scale enterprise software purchases.

Often the definition of the target market and its segments does not lead to a static result, but needs to be considered over the strategic timeframe. In particular in the Powerboat or Icebreaker phase, the initial focus may be on a rather narrow market which can be widened over the strategic timeframe. So it makes sense to look at the evolution of the market definition as part of the roadmap.

3.4.5 Outcome and Impacts

A good market and segment definition is of key importance for the product strategy. It is the basis for product definition, positioning and value definition, forecasting, channel and partner selection, and all the marketing work. At any point in time, there needs to be a clear definition that is accompanied by results from market analysis (see Sect. 5.5):

- Market scope and boundaries.
- Appropriate segmentation.
- For each segment:
  - Size (in annual revenue and/or annual number of licenses/customers).
  - Characteristics of customers.
  - Competitive situation.
  - Ease of adoption of your product by the segment in question.

This definition is tightly linked to product definition and positioning including customer value proposition (see Sects. 3.5 and 3.6) which are developed in an iterative process with market definition. It has an impact on requirements definition and selection (see Sects. 4.2 and 4.3) since the product contents need to be optimized for the selected market segments. Pricing is impacted by the competitive situation, and also by special characteristics and requirements in selected segments (see Sect. 3.10). And of course, the market definition is the basis for the work of Marketing and Sales (see Sects. 6.3 and 6.4).

3.4.6 Summary and Conclusions

The importance of the market, its definition, evolution and ongoing analysis cannot be overestimated. A market can be understood as a set of sets of potential buyers, potential sellers, goods and services, respectively, at a particular point in time. The success of your product depends on its fit to the target market. The better you understand the relevant market and its segments, the more you can improve that fit.

3.5 Product Definition

3.5.1 Overview

The product definition describes the product on a rather abstract level, i.e. what is it, and what is it not, over the strategic timeframe. The definition addresses users, intended use, functionality, quality, user experience (UX), compatibility, technical constraints, customization, delivery model, and the whole product offering. All of them may change over time.

This section describes the contents of a product definition and outlines approaches how to get to a good product definition. The reader will understand the related processes and the impacts.

3.5.2 Concept

As part of product strategy, the product definition details the product vision without turning into a full-blown specification. It is intended to give guidance to the product team whenever decisions are required what is inside or outside of the product scope, in particular with regard to requirements. It is also intended to provide a high-level product description that can be used as a basis for marketing material or for educating new team members.

Intended use and users provides rough descriptions of how and by whom the product is intended to be used. While the detailed definition of use cases is part of the requirements work, and the detailed definition of personas is part of UX design, here only high-level descriptions or lists of roles are provided. For example, when IBM defines a new database management tool, they can simply say it is intended for database administrators and system administrators and their administration tasks.

Functional scope describes which functional areas are to be covered by the product, and which functional areas are outside of the product's intended capabilities. The latter is very helpful in the area of product requirements management (see Sect. 4.2). For example, SAP started out with Enterprise Resource Planning (ERP) functionality, but added data warehouse, data analytics, or CRM (customer relationship management) functionality at later stages of its life cycle.

Quality scope describes in which ballpark the quality attributes of the product are intended to be. While the detailed definition of quality requirements is part of product requirements management (see Sect. 4.2), here the ballpark is often determined by listing comparable products or product areas. For example, when Microsoft defines a new version of Office, they can define the quality scope by saying that quality needs to be at least as good as with the previous version, but 20% better in the area of technical performance.

User experience (UX) design scope describes in which ballpark UX is intended to be. Similar to quality scope, the detailed definition of UX requirements is part of product requirements management (see Sect. 4.2) in combination with UX design work. Here the UX ballpark is often determined by listing comparable products or

product areas. For example, when a company defines a new product, the product manager can define the UX design scope by saying that UX measures need to be at least as good as with a particular existing product in the company's portfolio, but 20% better in the area of subjective satisfaction, i.e. user happiness.

Compatibility scope is relevant for new versions and releases or follow-on products of existing products. It describes to which degree compatibility is intended. By upwards compatibility, it is understood that:

- In changing from software version n of a product to the next version n + 1, existing functions of version n continue to be supported.
- Data from version n can be transferred to and used with version n + 1 without changes.
- Interfaces of version n (APIs, Interfaces for other Systems/Products) remain unchanged.

Should only parts of these conditions be fulfilled, we speak of function, data and interface compatibility. Full upwards compatibility makes the customer's migration from version n to version n + 1 smooth and inexpensive. However, frequently with a new release of a product comes an expansion and change of the underlying data model which leads to changes to the data structures. In this case, data compatibility cannot be achieved easily. A separate data migration is required, for which the software vendor should preferably provide in the form of procedures and scripts.

By downwards compatibility, it is understood that:

- Data from version n + 1 can be transferred to and used with version n without changes.
- Version n + 1 can communicate to version n, i.e. version n interfaces are supported.

In contrast to the upwards compatibility, downwards compatibility cannot always be expected or presumed. For example, when Microsoft introduced the new .docx format for Word documents with Word 2007, .docx documents could not be read with older versions of Word. In order to address that problem, Microsoft made a Compatibility Pack available for free which enabled some older Word versions to handle .docx documents.

Technical constraints are restrictions that are decided upon early. The technical constraints complement the quality scope of the software product from the technology, platform, and deployment perspectives. The term comes from requirements engineering, but these early decisions need to be made as part of the product definition. For example, a software vendor may decide that a particular software product will only run on Android and iOS as mobile operating systems, but not on Windows 10 Mobile.

Customization scope describes to what degree customization options are intended to be provided. The range goes from standard product as it is, i.e. no customization, to customization options for each and everything. With no

customization, you run the risk that quite a number of your target customers do not buy the product because it deviates from their respective requirements in a few important areas. With too many customization options, you spend a lot of development effort on customization, and it creates a lot of effort and cost for customers before they can become productive with the software product. With B2B products, customization options are typically more extensive than with B2C products. For example, most smartphone apps do not provide any customization options in order to keep things as simple as possible.

Product lines are a concept that gives the vendor more variability in adapting the product to different product environments (see Sect. 2.2) while customization refers to what a customer can do to adapt a product to his environment.

Delivery model describes in which way a software product is made available to customers. The alternatives are licensed product vs. Software-as-a-Service (SaaS) offering or a hybrid of both. Historically, the standard delivery model for software vendors has been that when a customer buys a license for a software product, the software is transported to the customer on a medium like CD, the customer installs the software on his computer hardware (possibly with the help of the vendor) and runs it on his own. Today, it is often downloaded over the internet, e.g. on markets for apps, components, or access to software or ICT services. Depending on the type of contract, the customer may be entitled to maintenance updates that he can install on his computer over time. Over the last years, the Software as a Service (SaaS) delivery model (see see Sect. 2.2) has emerged that often comes with pricing models like subscription-based pricing or funding through advertisements.

The reason why we discuss these delivery and pricing models here is that they can have significant impact on the specification and implementation of the software product. If SaaS is chosen as the delivery model it means that the software must be highly scalable and capable of multi-tenancy. These are requirements that a lot of standard license software products do not fulfill. The same is true for a pricing model like usage-based pricing that cannot easily be implemented on top of an existing piece of software, but needs to be considered as a requirement early on in the initial design.

Whole product offering stands for the complete set of elements that are to be offered to the customers in order to provide a complete solution (see Sect. 2.4). This set can contain not only the software itself, but also additional software components and professional services from the vendor's portfolio or from other vendors. We have experienced situations where two customers bought the same product, but got different sets of components delivered. Therefore, we recommend that product definition encompasses the definition of the whole product offering including a **complete list of components** that belong to the standard product offering and are included in its price. Since product-related human services are part of the whole product offering, the service strategy needs to be linked to product definition (see Sect. 3.7). For software components from other vendors, gaps in the offering need to be identified that can be addressed by those components. The selection of other vendors as partners has to be synchronized with ecosystem management (see Sect. 3.11).

3.5.3 Defining a Product

Product definition is an iterative process that is intended to lead to consensus between the relevant stakeholders, i.e. Product Management, Marketing, Sales, Development, Services, and executive management. The result is usually not static, but evolves over the strategic timeframe and beyond.

The iterative process includes:

- Consultation of stakeholders and contributors.
- Development and testing of alternative product definitions with stakeholders and customers.
- Elaboration in cooperation with selected stakeholders.
- Roadmapping for refinement and buy-in (see Sect. 4.1).

The product manager needs to find a balance between continuous learning and improvement that translate into changes of the product definition, and some level of stability that provides a basis for customer communication and avoids wasted efforts.

There are a number of interdependent factors to be considered: target market segments, customer value and positioning, service strategy, budget, pricing, and the product's life cycle phase (see Sect. 4.4). For new product development, i.e. in the Powerboat and Icebreaker scenarios, product definition means a rather dynamic learning process. During later phases of the product life cycle, the product definition tends to be more stable.

For new products, this is directly related to the minimum viable product (MVP, see Sect. 2.5) which Eric Ries defines as "that version of a product that enables a full turn of the Build-Measure-Learn loop with a minimum amount of effort and the least amount of development time" ([Ries11], p. 77). This focus on learning may be helpful for a startup, but from a business perspective, Steve Blank's definition of the MVP as the minimum feature set "that some customers are willing to pay for in the first release" [Blank10] seems more on the point. He sees the MVP as a tactic to reduce engineering waste and to get product in the hands of early adopters soonest. Of course, that will lead to learning. We define

Minimum Viable Product (MVP) = The minimum feature set of a new product that is derived through a learning phase and that some customers are willing to pay for in the first release.

3.5.4 Outcome and Impacts

The product definition is usually documented as text. When the organization decides to document the strategy in a formal product strategy document, product definition is part of it. Product definition is an important element for positioning (see Sect. 3.6) and sets the scope and the boundaries of the product for the whole product team.

3.5.5 Summary and Conclusions

Product definition is an important activity for understanding what the product is about at any point in time, and how focus and priorities change over time. It covers intended use and users, functional scope, quality scope, UX design scope, compatibility scope, customization scope, and the delivery model. The Whole Product Offering needs to be considered. Developing the product definition requires an iterative process that involves all relevant stakeholders.

3.6 Positioning

3.6.1 Overview

With positioning, an approach is defined for communicating a product to potential customers. This includes an analysis of customer value, often differentiated by market segments, and the selection of sales channels through which target customers can be reached in a profitable way.

Concise, understandable positioning is a key product success factor. The objective is for the product "... to occupy a clear, distinctive, and desirable place relative to competing products in the minds of target customers." [KotArm15]. It makes it easier for sales and marketing to address customer groups with the right messages. It also helps internally in making all product-related decisions.

One of the important aspects of positioning is the question how a product differentiates itself in the market. This question is related to the concept of Unique Value Proposition which describes value elements that none of the available alternatives can provide. For proprietary products, the typical differentiating arguments have been better functionality, higher level of integration, performance etc.

In some markets standardization is considered equally important. But how can a product differentiate itself if it only implements a public standard? The vendors found this answer in the 1970's and still apply it: They implement the standard so that they can claim standard-conformance, and then they add useful proprietary features. Once a customer makes use of these features, he is tied to the vendor's product. So the features not only provide differentiation, but also the glue that makes a product more difficult to replace.

Positioning can also be relevant within the product portfolio of a software company, in particular when there are products with overlapping functionality. Such an overlap can easily create confusion for potential customers, the own sales force, and channel partners which needs to be addressed by clearly positioning the products against each other.

3.6.2 Customer Segmentation and Value Proposition

Once the market for the product is defined (see Sect. 3.4), the positioning must focus on describing the value of the product. Again this has to be considered over time since the value of the product will hopefully increase with each new version or release. The value definition needs to be approached from a customer perspective, e.g. what is the business value that the customer will get from using the product? Customer segmentation, i.e. building market segments as subsets of the total market of the product, may be needed when different segments experience different business values from using the product.

The Business Model Canvas can be a helpful tool here. Whether evolving an existing business model or creating a new one, the business model generation process often starts with the two canvas segments called "Customer Segments" and "Value Propositions". These two segments are closely linked through the underlying concept of what customers need or want, specifically their pains, their desired gains, and their "jobs to be done".

Product managers need to be specific about the customer segment(s) they are targeting so that they can develop a thorough understanding of customer needs. This understanding in turn is key to developing compelling value propositions that help sell the product. Here methods like the Value Proposition Canvas can help [OstPign14].

However, this does not mean that customer segments always have to be very narrowly focused—according to Moore [Moore14, Moore04] it depends on the maturity stage of the respective market how broad or narrow the target segment can be. For example, when bringing to market a completely new type of B2B product—what Moore calls a new product category—it is often useful to kick-start mainstream adoption by focusing on a small, well-defined market niche (the beachhead segment). The strategy is to expand into adjacent market niches later—one after the other (bowling alley strategy). Once the new product category is better understood in the market and broad adoption sets in at a fast pace (tornado phase), the vendor's top priority needs to be capturing market share. In this stage, an undifferentiated strategy is suitable.

The customer understanding is translated into the value propositions that (hopefully) strongly resonate with the customer segments and help sell the products and services [OstPign10]:

- Pain relievers "eliminate or reduce negative emotions, undesired costs and situations, and risks your customer experiences or could experience before, during, and after getting the job done".
- Gain creators "create benefits your customer expects, desires or would be surprised by, including functional utility, social gains, positive emotions, and cost savings".
- Products and Services lists the products and services that together deliver the value proposition.

It is not easy to determine what the drivers of customer value are for a certain product category. Anderson and Narus [AndNar98] suggested generating a list of value elements and building customer value models. Almquist e.a. [AlmSenBl16] build an "Elements of Value Pyramid" based on Maslow.

Especially in B2B technology markets, including B2B software, it is very common that core products require add-on products, integration with partner products, and accompanying services as a minimum configuration to deliver a compelling value proposition. Geoffrey Moore [Moore14] emphasizes the importance of this—he calls this the whole product concept. Based on the value propositions, software product managers need to identify which additional products and services are required to deliver the whole product, and they must make sure these crucial whole product components are actually available to customers—either from the vendor itself or from partners. This needs to be reflected in the Service Strategy. Partners that contribute important whole product components need to show up in the canvas, either in the channels section or in the key partners section.

3.6.3 Channels

Directly related to the definition of the target market (see Sect. 3.4), the product definition (see Sect. 3.5), and the positioning of the product is the question of sales channels. Channels define the path through which goods and services as well as the compensation are transferred between vendor and customer. We define

Channel = A sequence of intermediaries through which goods and services as well as the compensation are transferred between a company and its customers.

With this definition, an internal direct sales force is included in the intermediaries. What is the best way to reach potential customers in the selected target market segments? That can be a direct sales force, and/or indirect sales through partners or the internet (see also Sects. 3.11 and 6.4).

There are two criteria that influence the channel selection the most. One is access to target customers, the other one is channel cost in relation to the financial picture of the product, in particular price which may be related to the selected delivery model (see Sect. 3.5). For low-price products, the cost of a direct sales force is usually not bearable. On the other hand, for high-price products in a B2B market, customers often expect a personal contact with a sales representative. In a lot of markets, software vendors cannot easily get direct access to target customers, but are dependent on partners who have established relationships with those target customers or the local proximity that allows less costly access. When a vendor uses multiple channels in parallel, there is the risk of channel conflicts, i.e. channels competing with each other. This needs to be anticipated and managed (see Sect. 6.4).

There are two main categories of sales channels for software products: The physical (human) channel and the virtual channel. In the physical (human) channel, direct sales means that products are marketed directly by the company to customers, eliminating the need for an external intermediary (e.g. wholesalers, advertisers or

retailers). An example is a direct sales force, where representatives have in-person contact to customers. Key account managers and key customers come together ("farming") and new customers are recruited from the defined target market ("hunting"). Since the customers no longer seek information technology as such, organizations should provide IT solutions that improve the business and describe advantages and positive effects. Direct selling means that an IT sales specialist communicates the resulting financial benefits. For global players it can be useful to implement a "global account management" to be present wherever the customer is.

Another physical (human) channel is indirect sales which means that selling of products is conducted by partners, e.g., companies such as retail shops or wholesalers. For the different types of sales partners see Sect. 3.11. The advantage of the indirect sales channel is that more potential customers can be reached through the partners and their customer relationships. Decisions are required whether a one- or multi-tier partner model is appropriate and how channel conflicts can be avoided with a multi-channel approach. On an operational level, the management of the selected partners is part of ecosystem management.

Telesales (or inside sales) means that a sales person contacts customers to sell software products, either via telephone or through a subsequent face to face or web conferencing appointment scheduled during the call. Telesales is still being used even though it has increasingly been considered as an annoyance by many customers.

Internet sales belongs to the virtual channels. Here software products are sold on the web through e.g. web-based apps, app stores, etc. This works with both license products, SaaS and advertising. Customers buy conveniently and easily from the comfort of their home or office at any time.

A further classification of channels differentiates

- Free versus paid: *Free channels*, e.g. social media or blogging, contain inherent costs (non-zero human capital cost). In contrast, paid channels like search engine marketing require explicit investment.
- Inbound versus outbound: *Inbound channels* rely on being found by the customer (pull messaging, e.g. blogs, e-books, and webinars), while *outbound channels* reach for customers (push messaging, e.g. trade shows, cold calling).
- Automated versus direct: *automated channels* to reach a high number of customers at low cost, versus direct channels (see above).

A software company must make a fundamental decision regarding its channel mix: should the various sales channels be allowed to overlap and compete with each other or should each have a well-defined target market and objective? Sales distribution channels that are to overlap each other are measured by common objectives in order to avoid channel conflicts in so far as possible. This requires objectives which are higher to reflect the sum of what would have been the individual objectives. This may result in the individual channels all pursuing the large opportunities rather than going after the target market which they were intended to pursue. The alternative, assigning specific target markets and

objectives, may lead to some squabbles at the boundaries, but is often adopted in an effort to focus sales more strongly and avoid unnecessary overlap-ping. Channel conflicts are almost always counterproductive in practice.

3.6.4 Process

Positioning is an iterative process that is intended to lead to consensus between Product Management, Marketing, Sales, and executive management. At least Marketing needs to participate in this work. The result is usually not static, but evolves over the strategic timeframe and beyond. There are a number of interdependent factors to be considered, in particular target market segments, product definition, and pricing.

Determining customer value and how it is generated requires some in-depth analysis how and for what customers are going to use the product, and what their alternatives are. Alternatives can be competitive products, custom solutions, or doing nothing. If possible, it can be helpful to involve existing or potential customers in this analysis. The drivers or parameters need to be determined that influence customer value. Ideally, the relationship between the parameters and customer value is expressed in a formula. Customer value can be considered in absolute terms, or in terms relative to the identified alternatives that a customer has, or relative to the price of the product.

For each of the candidate channels, the potential for access to potential customers in the target market segments and the resulting cost in relation to the financial model of the product need to be analyzed. The results are the basis for channel selection.

The sales channels a company uses for a particular product are selected according to specific criteria:

- Target market (segments) (see Sects. 3.4 and 3.6.2): Which channel has the highest potential to reach customers in the selected target market?
- Product definition (see Sect. 3.5) including the definition of the "Whole Product" and the delivery model: Can partners who provide components act as channels?
- Sales cost: What are the cost and benefits the selection of a particular channel entails.

The cost of direct sales is usually higher than the cost of indirect sales channels. Therefore, the higher-priced direct sales tends to be restricted to software products in the high price segment. Software products at lower prices are rather distributed through indirect channels. In addition, other factors should be considered as part of a comprehensive channel strategy:

- The relationship frequency [some channels are used systematically and repeatedly, others opportunistically (one-offs)].

- The place of purchase (online retailer versus software retail store around the corner).
- The strategies in software ecosystems, i.e. niche players, keystone players, dominators (see Sect. 3.11).
- The purchase frequency or the degree of willingness to buy (impulse buyers versus regular customers).
- The purchase occasion.
- The attitude towards the product or the service.
- The use rate.

The selection of channels usually results in a mix that can include direct sales including telesales and virtual sales, and partner sales. The terms and conditions for the channels need to be defined such that channel conflicts are prevented. The operational responsibility of managing this marketing mix on an ongoing basis is with Marketing.

3.6.5 Outcome and Impacts

Customer value, the advantages that a customer experiences when using the software product, can be described in absolute and relative qualitative terms. For example, we can identify a certain feature as highly valuable for customers in a particular segment and show this feature to be a differentiating factor compared to the alternatives. This qualitative description provides valuable input for the development of marketing messages by Marketing.

Simply evaluating the value as high, medium, low for the different market segments may be a good start that shows how well the product fits the needs of the respective market segment. It can also be used for a rough comparison with the alternatives. However, it is not good enough as input for marketing and pricing.

The drivers of customer value need to be identified on a more detailed level. They depend on the type of software. For transactional systems like an airline reservation system, the driver can be the number of transactions, i.e. flight reservations, or the revenue made through the system. For productivity tools, it can be the number of users or the usage time, or the savings in terms of time that employees need for particular tasks. The parameters identified to drive value are the ones that can be used to calculate the price in a value-based way (see Sect. 3.10). For a potential customer, the relationship between a product's value and its price in comparison to the alternatives determines the buying decision.

The definition of the channel mix may change over time based on experience, or when market segments are added to the target market definition. For new product development, i.e. in the Powerboat and Icebreaker scenarios (see Sect. 2.5), positioning means a rather dynamic learning process. During later phases of the product life cycle, the positioning tends to be more stable.

3.6.6 Summary and Conclusions

Positioning provides important information how the product is going to be communicated to the market. The core of positioning is the value proposition. Differentiation by customer segments may be required. In order to reach the customers in the segments in a cost-effective way, sales channels have to be selected and implemented. Positioning is highly relevant for Marketing and Sales, but also for other product management activities like pricing or release planning.

3.7 Service Strategy

3.7.1 Overview

The software product manager is supposed to manage the whole product offering which includes product-related human services (see Sect. 2.4) even though the responsibility for the provisioning of these services may organizationally be separated from the software product organization. The services are needed to enable customers to become productive with the product as fast as possible and stay productive as long as they use the product.

This section describes the contents of a service strategy. The reader will understand the related processes and the impacts.

3.7.2 Concept

Product-related human services can include education, installation, customization, operations, and maintenance including a user help desk covering technical and non-technical problems. Typically, these services are priced separately, sometimes they are bundled with software product offerings. Details of these services are described in Sect. 6.6.

A software vendor needs to decide if and to what degree these services are provided by the vendor itself, and/or through partners. Resulting revenue from self-provided services and capacity considerations are relevant aspects. Also, partners who provide product-related services are also motivated to sell the product to their customers. On the other hand, services can help the vendor to intensify the customer relationship, and to learn about the customer environment and how a customer uses the product. This information is valuable input for the positioning and evolution of the product. So the question who provides which services is clearly of strategic importance. Communication between the in-house service team and Product Management and Development is usually easier when they all belong to one product organization.

Another type of human service is custom software development by which customer-specific functionality is added to the standard software product. This becomes relevant when a customer has requirements that the software vendor

refuses to implement in the standard product because they are too customer-specific, i.e. not of value to other customers. When the vendor decides to offer this kind of custom software development, it needs to be clearly separated from the standard product both organizationally and contractually. In particular, for cost reasons this custom code must not be covered by the maintenance contract for the standard product. There can be additional customer-specific services like system integration.

For new product development, i.e. in the Powerboat and Icebreaker scenarios (see Sect. 2.5), services for the first customer are often provided out of Development since that first customer is intended to provide domain knowledge and the vendor organization needs to learn how to provide those services. The services for later customers, i.e. after the product is launched, needs to be done by the units declared as responsible in the Service Strategy including external partners. During later phases of the product life cycle, additional services may be identified and added to the service strategy. During the maturity and decline stages of license products, maintenance revenue usually plays an increasingly important role in the financial picture of the product.

3.7.3 Process

Defining the service strategy is a process that is intended to lead to consensus between Product Management, Services, Marketing, Sales, and executive management. At least Services and Marketing need to participate in this work.

In order to define the whole product, product-related services need to be part of the product definition (see Sect. 3.5). For the service strategy, the product manager needs to take a number of aspects into account:

- Which product-related services are needed for offering a complete solution that satisfies target customers' needs, and that target customers ares willing to pay for?
- Who can provide those identified services with regard to capacities, skills, and customer acceptance?
- What is the right balance between revenue generation and learning from self-provided services, and customer access and potential additional product revenue through external service partners who also act as sales partners?

Once the service strategy answers these questions, it is the task of either the internal Services unit or ecosystem management for partners to make sure that the identified services are available at the time when they are needed.

3.7.4 Outcome and Impacts

The service strategy is documented in text that is usually part of the product strategy document. It lists the services identified as part of the whole product offering and defines who is supposed to provide these services. If additional services outside of the whole product offering, but related to it, are planned, like custom software development, these are documented as well.

Product-related services can have a significant impact on customer satisfaction. They may also contribute significant revenue. The service strategy gives direction to Services as well as Ecosystem Management.

3.7.5 Summary and Conclusions

Product-related services are important for the business success of the software product. That is why the service strategy needs to be integrated into the product strategy and tightly linked with product definition. It looks primarily at product-related services like education, installation, customization, operations, and maintenance that help a customer to become productive with the product, secondarily at additional customer-specific services like custom software development and system integration. The service strategy answers the questions which services are needed, and who is going to provide them. The product manager needs to manage the product-related services as part of the Whole Product offering.

3.8 Sourcing

3.8.1 Overview

Sourcing is the process of ensuring that the resources required to implement a product strategy are available whenever they are needed. Based on a corporate sourcing strategy that defines if, for what and to what degree external resources can be used, decisions on a software product level focus on human resources, make-or-buy for software components, and use of external services and data sources.

This section describes the criteria for sourcing decisions, related processes and impacts.

3.8.2 Concept

For sourcing, there are basically two options: Required resources can either be internal, i.e. come from inside of the company, or external, i.e. from outside of the company. So once the positioning and definition of the product for the strategic timeframe has been done, the activities need to be identified that are required to implement the product strategy including very rough estimates for efforts and skill

and time requirements. These estimates can be refined on an ongoing basis, but initially they are usually good enough to develop a first idea of resource requirements.

For software products, the most important resource is skilled human beings. They can either be employees of the software organization, or hired from outside of the company. In bigger organizations these decisions usually need to be aligned with corporate resource management (see Sect. 5.4), e.g there may be a sourcing strategy on the corporate level that gives guidelines as to if, in which areas, under which circumstances, and to what extent the company wants to use external human resources. Under these guidelines, a product team can develop a product-specific sourcing approach.

There are a number of reasons why a software organization may want to work with external people, e.g. special temporary skill requirements, temporary capacity requirements, a general wish for resource flexibility in case of changes of strategy and plan, or cost considerations.

Since the mid-1990's, a lot of companies have worked with external people in locations with a considerably lower cost level. This is known as near-shore or off-shore outsourcing. In the software area, some companies have managed to make this work and save money, but in a lot of cases the hidden cost generated by communication overhead and lost time due to communication problems turned out to eat up the calculated savings. There can also be risks from skill dependencies. A lot of literature is available on outsourcing, e.g. [OshKoWil11].

In some situations, it can make more economic sense for a software organization to integrate a software component supplied by an external partner than to develop that component in-house. This is known as the Make-or-Buy decision. It may also apply to complete products. Reasons to buy can be faster time to market, improved quality, skills shortage, or resource shortage. Aspects like interoperability, terms and conditions, cost, ongoing development and support, and setup of the cooperation need to be considered. The product manager should not leave this decision to development since developers usually want to make and not buy. In case of a buy decision, it is of utmost importance to negotiate the contract with the partner that provides the software such that the dependence on the partner does not lead to a disaster like losing the right to use the software on short notice, or facing extreme price increases. There should also be consideration of what could happen in the longer term. Because IBM was not sure of the future of the PC, they refused to pay Microsoft for PC DOS and insisted on a royalty arrangement. The cash flow generated by that decision enabled the emergence of one of IBM's strongest competitors.

One option is the use of open source software. Here, the product manager needs to assess the viability and ongoing support based on the level of activity in the respective open source community, and the legal risks due to the type of open source license (see Sect. 3.12.4). Again, we do not recommend to leave this decision to the developers alone since they are usually not aware of the legal implications.

Make-or-Buy decisions are also relevant in areas like infrastructure, or data. A good example is hosting for SaaS offerings. Reasons for partnering can be that a software organization has no experience with hosting for customers, or that there may be advantages in terms of price and scalability due to the economies of scale that a big hosting provider achieves. An example for sourcing data is stock market data.

3.8.3 Decision-Making for Sourcing

Given the business impact and risks that are associated with sourcing decisions, the product manager always needs to be involved in the decision making and selection of sourcing partners. Within the limitations defined by the corporate sourcing strategy, a product organization needs to decide if it can implement its product strategy with the available internal resources, or if and where external sources need to be added. That decision is usually not made by a product manager alone, but needs to be driven by Product Management. Once a decision for external sourcing has been made, an external partner has to be selected. Criteria can be financial stability and reputation of the partner company, skills of the available people the partner company can provide, cultural fit, terms and conditions, in particular price, or—in case of software components—functional and technical fit of the partner's component to the requirements. Contract negotiation may fall under a product manager's responsibility unless there are specialized units like partner management (see Sect. 3.11). For service contracts, service level agreements need to be negotiated as part of the contract (see Sect. 3.12).

The selection of a partner company can be a trigger for analysis if the acquisition of that partner company could bring a strategic advantage, e.g. in terms of product set, skills, market share, risk reduction or financial impact, and is financially doable. However, the process of analysis, due diligence and acquisition is outside of the product management scope. Bigger companies have specialized merger and acquisition units for this (see [Popp13]).

The responsibility for the implementation is usually with the organizational unit whose resources are extended by the sourcing decision. That can be Development for software developers or the integration of an externally sourced software component, Quality Management for testers, or Operations for a hosting partner. Though typically not responsible for the implementation and operational management, Product Management is recommended to monitor the activities on a frequent basis in order to take corrective actions in case of negative business impacts.

3.8.4 Summary and Conclusions

Clever sourcing decisions can shorten time-to-market, increase quality and/or improve financial performance. However, external sourcing comes with increased risk that needs to be managed well. It is part of the entrepreneurial responsibility of

product managers to drive these sourcing decisions, whether they concern human resources, or make-or-buy decisions for commercial software, open source software, infrastructure, or data.

3.9 Business View

3.9.1 Overview

The primary objective of product management is to achieve sustainable success over the life cycle of the product (or family or line). This generally refers to economic success, which is ultimately reflected by the profits generated. Since profits lag behind investments, i.e., an investment phase involving unreimbursed costs will be followed by an extended profitable phase, a longer-term perspective is appropriate. Therefore the product manager has the role of a mini CEO who has to plan and keep track of the business aspects. This is not limited to the product manager keeping track of the revenue numbers (see Sect. 3.13). It means a much more active role in shaping the parameters that are paramount for the economic success of a product or product family. The relevant topics are covered in this section, plus pricing in Sect. 3.10 and performance and risk management in Sect. 3.13.

3.9.2 Business Case

Since a product manager is responsible for the economic success of his product over its life cycle, the business aspects play a major role in all of the product manager's tasks and activities, from the analysis and prioritization of individual requirements to the positioning of the whole product. We define

Business Case = A decision support and planning approach for comparing the costs and benefits associated with a proposed initiative.

Depending on company rules, a business case may be required for each requirement within the requirements management process (see Sect. 4.2). At the minimum, a business case is required for any new release of a product: if an investment of &3M is proposed for a release there had better be a forecast of additional revenue from new customers or increased product usage which is sufficiently large over the vendor's return on investment (ROI) period to give the company their desired profit margin. The calculation of the cost side bears some risk, but is typically easier than the value side; for one thing, the costs all occur within the developing company's control. On the value side the challenge is threefold: often benefits and value are of a qualitative nature that is difficult to convert into actual earnings or savings; the monetary benefits may (and usually do) vary from customer to customer; and the benefits/value may be realized over an extended time period after the release is made available. A software value map can be helpful by which value can be analyzed in a systematic way [KhuGorW13].

To be effective, the business case should communicate the following information:

- The description of the undertaking (name, unique identifier, short text).
- The underlying assumptions.
- An estimate for the required investment (in absolute or relative numbers).
- The approach to generate business benefits with estimates for earnings over time (in absolute or relative numbers).
- A risk, sensitivity, and contingency analysis.

When business cases are used for selecting or prioritizing alternatives, relative numbers may be sufficient. When absolute numbers are used, there is considerable uncertainty and risk that needs to be addressed. This can be done by looking at different scenarios (e.g. best, median, worst) and analyzing the risks, the sensitivity of the estimates to the risks, and contingencies, i.e. possible actions to mitigate those risks. Given the high level of uncertainty, some people in the software industry claim that any business case is more a pseudo-scientific quantification of the gut feelings of the decision makers than a valid prediction. But even if this were true there is value, in particular from planning the approach to generate the identified business benefits and from the contingency planning.

A business case is credible when it is complete, balanced with important scenarios elaborated, and with underlying assumptions made explicit and accepted by all stakeholders. There is a lot of literature on business cases, e.g. [SheeGall15] or [Schmidt02].

With business cases, a continuous improvement process is highly recommended. Each business case needs to be revisited after an appropriate period of time so that the actual outcomes can be compared to the business case and deviations can be analyzed. That helps the organization to learn and improve.

3.9.3 Business Plan

The business plan documents a complete picture of the financial numbers for the product over its strategic timeframe. It is often also called financial model. Business planning is a process that aims at getting the relevant stakeholders' consent to the business plan. The business plan is a forecast regarding cost and revenue that leads to a plan for resources and budgets. We look at forecasting in Sect. 3.9.4, at the cost side in Sect. 3.9.5, the revenue model in Sect. 3.9.6, and pricing in Sect. 3.10.

We recommend to explicitly document any assumptions that the business plan is based on. Since it usually gets a lot of attention and scrutiny from top management, product managers are advised to be prepared to explain each and every detail of the business plan.

3.9.4 Forecasting

Estimates and forecasts can be quite good under certain conditions: stable environment and historic data that can be simply extrapolated. Unfortunately, we often have to come up with forecasts when those conditions are not present. That is what Philip E. Tetlock and Dan Gardner's book "Superforecasting—The Art and Science of Prediction" [TetGar15] is about. However, if you are seeking a recipe for reliable forecasts the book will disappoint you. The majority of the content concerns Tetlock's huge research studies and their findings. There is clear evidence that some people are consistently and significantly better than average at this kind of forecasting. And there is a section on more practical advice called "Ten Commandments for Aspiring Superforecasters", but it is only an appendix and comes more as an afterthought than as the climax of the book. It lists important factors that have an impact on the predictive quality of forecasts:

- Triage: Focus on predictable questions, not trivial and not unsolvable ones.
- Break seemingly intractable problems into tractable sub-problems.
- Strike the right balance between inside and outside view.
- Strike the right balance between under- and overreacting to evidence (when you adjust your forecast iteratively).
- Look for the clashing causal forces at work in each problem.
- Strive to distinguish as many degrees of doubt as the problem permits but not more: look at degrees of uncertainty.
- Strike the right balance between under- and overconfidence, between prudence and decisiveness: trade-off between accuracy and finite decision time.
- Look for the errors behind your mistakes but beware of rearview-mirror hindsight bias: do thorough post-mortems.
- Bring out the best in others and let others bring out the best in you: teamwork increases success statistically.
- Master the error-balancing bicycle: only forecasting practice and feedback loops can improve your forecasting ability over time.
- Don't treat commandments as commandments: There are no firm rules that apply to any case.

As you can see from the last bullet, this is half-hearted advice. We cannot do without forecasting, but it remains more art than science.

3.9.5 Cost Structure and Management

For software products, development effort is usually the largest determinant of cost. So in order to manage cost, an upfront estimation of the development effort is needed. The discipline of software engineering has come up with a number of estimation methods over time, e.g. the more recent Cost Estimation, Benchmarking, and Risk Assessment (CoBRA method) described by [Trendow13]. These methods

are usually targeted at custom software development projects, but can easily be adapted to software product development projects. However, all of them use historical data and experiences as a base to estimate a new undertaking. When there is no history the reliability of estimates is typically low (see Sect. 3.9.4). Planning Poker is a technique that works quite well for effort estimation (described in Sect. 4.3.3).

Costing or cost management means that the target cost for an undertaking is defined, and then the undertaking is managed against that target cost which is often called a budget. That cost management is typically delegated to the responsible line or project manager.

As a major part of the financial model of a product, we need to understand the cost structure of that product in a broader sense. A financial model for a software product is typically structured similar to the income statement (also known as profit and loss statement, or P&L) for an entire company.

In particular, for software products, the income statement is typically structured as shown in Fig. 3.5, with costs broken down into several categories. The first category, cost of revenue, is highly variable. Apart from cost of revenue, we usually distinguish between four other categories of operating expenses that are "fixed", i.e. they will not move immediately in sync with a short-term revenue spike or revenue decrease. These four fixed cost categories are: research and development (R&D), sales and marketing, general and administrative (G&A), and other operating expenses (such as asset depreciation).

Figure 3.5 shows the structure of a full income statement for a standalone software company, see for example the annual or quarterly reports of companies

Income Statement – Cost Structure

	Revenue	The "top line"
–	**Cost of revenue** **or Cost of Goods Sold (COGS)**	Revenue-related costs: parts, labor, e.g. support engineers
=	**Gross Profit**	
–	**Other Operating Expenses**	• Research & Development (R&D) • Sales & Marketing • General & Administrative (G&A) • Other operating expenses, e.g. depreciation of assets
=	**Operating Profit**	Profit from ongoing operations
–	**Non-Operating Expenses**	Interest paid or earned, taxes, ...
=	**Net profit**	The "bottom line"

Fig. 3.5 Income statement

like SAP, salesforce.com, Adobe, Microsoft, or Google. For an initial assessment of the profit potential for a new product (e.g. for a startup) and for company-internal business cases, it is usually sufficient to stop at the operating profit. In the company-internal case, that's roughly equivalent to the contribution margin that the product contributes to the larger organization.

In most traditional "brick and mortar" business models, for example a manufacturing business or a retail operation, the cost of revenue (labor, parts, etc.) eats up most of the revenue, often in the 70% range. Only the remaining 30% of gross profit are available to cover the fixed cost listed above. This typically leaves rather small margins for operating profit and net profit. Service businesses, for example support or professional service organizations of software vendors, have a similar cost structure, since their main cost driver is the cost of people delivering the services, and this is classified as "cost of revenue" as well.

However, for pure software products, the income statement looks quite different: small cost of revenue often leaves a gross profit of 70% or more—and this is needed to cover high fixed cost, in particular in R&D. Usually, there are no expensive assets, so depreciation of assets is not a big concern either. Exceptions can come from investments in infrastructure and licenses from other vendors. In reality, many software businesses rely on a revenue mix that combines "pure" software revenue, e.g. from license sales, with revenue from support and professional services. Depending on the revenue mix, the structure of their income statement falls somewhere in the middle between the two extremes described above.

3.9.6 Revenue Model

Part of the business plan is the revenue model. According to [Popp11b], a business may combine multiple revenue models, and each revenue model can rely on multiple revenue streams (see Fig. 3.6). If multiple revenue streams are combined in a revenue model, this is called a hybrid revenue model. Hybrid revenue models are very common in the software industry: for example, the classic software license model often combines a license revenue stream with support revenues and revenue from product-related services, such as installation and customization services—these services are often called "professional services".

Revenue streams are characterized by the following attributes which we will come back to when we discuss Pricing in Sect. 3.10:

- **Compensator**: who provides the compensation?
- **Effect**: the type of compensation, including
  - Payment.
  - No compensation, e.g. usage of open source software.
  - Compensation in other goods or services, e.g. a Google user compensates Google for its search service by providing information about his areas of interest.

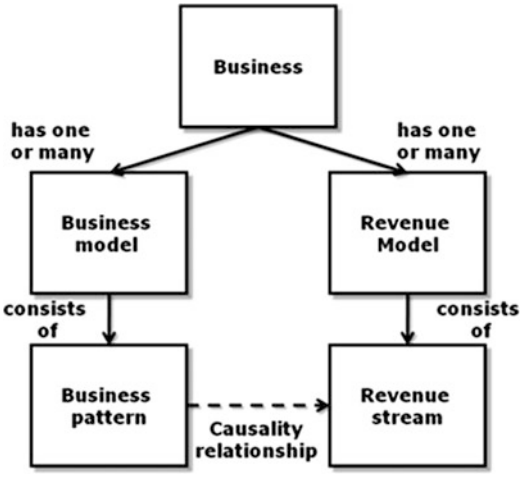

Fig. 3.6 Business model and revenue model [Popp11b]

- **Rating**: how is the consumption of goods or services measured?
  - Time-based, for example usage for 1 month,
  - Usage-based, e.g. gigabytes of storage, number of unique or concurrent users that are permitted to use the product,
  - Functionality-based, e.g. silver, gold, platinum editions with increasing functional scope.
- **Charging**: how is the compensation amount for a certain rating of goods and services determined?
  For example, charging a fixed fee per user per month for using the software. This also includes different options regarding the frequency of payment:
  - Recurring revenue from regular business: support services, rental models, IP licensing.
  - One-time revenue types from regular business: perpetual license.
  - One-off revenue (not from the regular business): selling the IP, spin-offs, ...

In order to come up with ideas for revenue streams, Osterwalder and Pigneur [OstPign10] suggest considering the following key questions:

- For what value are our customers really willing to pay?
- For what do they currently pay?
- How are they currently paying?
- How would they prefer to pay?
- How much does each revenue stream contribute to overall revenues?

Before building the revenue model, software product managers ought to work with Finance to understand which revenue recognition guidelines are applied in their organization: when exactly revenue can be recognized is especially tricky for

software, varies somewhat between different accounting standards (e.g. IFRS which is heavily used in Europe vs. US-GAAP), and also depends on charging details (e.g. one-time license charge vs. recurring charges for a SaaS offering or a support contract) and bundling (e.g. pure software license vs. software bundled with professional services).

Product managers are typically in charge of building a revenue model that projects future revenue streams for their product, usually as a spreadsheet. There are two common approaches: extrapolation from the past vs. bottom-up. Both approaches use the established pricing strategy (see Sect. 3.10), as well as other input data, such as

- Current user and revenue base (in case of an existing product).
- Results from market analysis (market size, trends, growth rates, ...).
- Planned sales channels.
- Planned investments in sales, marketing, customer acquisition efforts.
- Experience values regarding channel effectiveness, return on investment (ROI) of customer acquisition expenses, or historic sales ramp-up (adoption curves) for similar products.

Let's first look at the scenario of an enterprise software product in an established, "steady-state" market. The product is the market leader in terms of revenue share. The market itself has been growing at around 10% annually for several years, no major market disruptions are on the horizon and market forecasts from industry analysts agree that this growth pattern will continue for the next couple of years. The market has been very fragmented, but is consolidating, and the product has been gaining market share steadily, growing about 2% points above the market growth rate. The product still has "room to grow" by taking market share from its competitors, since its current market share is still well below 50%.

The company has managed to slowly drive up average deal sizes across channels, so that the historic revenue growth was achieved with only a small expansion in sales expenses. For the coming years, the company plans to continue on this trajectory regarding sales investments. The company will continue to update and evolve the product as it did in the past, and no pricing changes are planned.

Here we use the approach of extrapolation from the past. Future annual growth rates are modeled by taking future market growth rate projections from analysts and adding the 2% points that the product traditionally has "outgrown" the market. While building the revenue model this way is rather simple, it should be noted how many context factors from the past must continue to be true in the future for the revenue projection to hold water. If only one of these context factors changes—either in the market or company-internally (e.g. changes in sales approach and investment), the revenue model does not provide plausible guidance any more.

Of course, that approach does not work for entirely new products where no historic data is available. We look at the scenario of a completely new product in a product category that does not exist yet. If successful the product will establish this new product category. Here, it is tempting to start with the estimated market

opportunity and to assume that the new product will capture a certain percentage of the market within a certain time period, say 1% in the first 3 years. This is called a "top-down" revenue model—a faith-based approach that does not enable any meaningful discussion, analysis, or planning. We rather go with the approach of the bottom-up revenue model starting with a model of the customer base, which in turn is based on the sales and marketing strategy.

The model for the customer base may use the following inputs:

- Conversion rates along the customer acquisition process, for example for a web-based sales process.
- Other key metrics driving customer acquisition and retention, for example the "viral coefficient" where applicable (one new user on average draws in another x users (see [Ries11]) and churn rates (what percentage of the customer base do we loose in a given time period)).
- Planned investment levels in sales and marketing, e.g. number of sales representatives, online marketing budget, etc.
- Data on the effectiveness of channels, and of sales and marketing expenses, e.g. average revenue generated by a sales representative per year or effectiveness of customer acquisition expenses, such as online ads.
- Average deal size/order size (where applicable).

Once a model of the customer base has been built, pricing information can be plugged in to create a revenue model. Since price structure and price levels are likely to change in the early phases of new product development, pricing information should be treated as input parameters to the model so that it can be changed easily.

Of course, when starting with a new product, all the inputs into the revenue model—from conversion rates to channel effectiveness to deal sizes—are assumptions. The quality of these assumptions can be improved by using historic data from comparable products as an initial best guess. But they still remain assumptions. The Lean Startup movement emphasizes the importance of systematically validating all the assumptions that are critical to the success of a business model [Ries11]. As more and more inputs to the revenue model get validated, the plausibility of the revenue model increases.

Building a bottom-up revenue model requires more effort, but it provides a number of benefits: it can turn into a very helpful tool to assess viability of the planned sales approach, and to run what-if analyses for changes in pricing.

For example, for a product that is sold via the web channel, A/B tests may be used to study the impact of different price metrics, discounts, or price levels on buying behavior and conversion rates. Results from these tests can then easily be plugged into the revenue model to determine the financial impact of the pricing changes and changes in conversion rates.

Finally, the revenue model can also help detect inconsistent assumptions in the business model, especially regarding the interaction between sales channels and product structure: for example in a B2B business model that relies primarily on

license sales through a direct sales force with long sales cycles, each sales representative can close only a small number of deals per year. Therefore, the average deal size will have to be at least in the US&100,000 range.

3.9.7 Bundling

Another interesting business aspect is the bundling of products. We define

Product Bundle = Set of products that is sold as one product with its own price.

A bundle can contain products from one or more vendors. There are different types of product bundles:

- Pure: The products are only sold as part of the bundle, not individually.
- Mixed: The products are sold as part of the bundle and at least some of them also individually.

Bundling may have several different motivations:

- The bundle makes it easier for the customer to buy a complete solution and signals that the vendor has already taken care of a tight integration of the involved products. The vendor hopes for increased revenue and profit.
- The bundle is less expensive than the sum of the individual products and thus gives the customer the impression that he saves money by buying the bundle. Again the vendor hopes for increased revenue and profit since a percentage of the customers would not have bought all products in the bundle individually.
- The bundle combines market-leading products with products that are new or have strong competition. The vendor hopes for increased market share of the new or weaker products because for customers who buy the market-leading products anyway the bundled new or weaker products look like zero-cost add-ons.

The third case is a frequent cause for legal action if the vendor has a dominating position with his market-leading products. An example is RealAudio's fight against Microsoft for bundling the MediaPlayer with the Windows operating system. In the second case, the price motivation has been analyzed scientifically (see [Biering04]), but unfortunately the results are often only applicable in special situations.

If a product manager is responsible for a product family, he may consider bundling within his family, but in most cases bundling means that several product managers have to cooperate, maybe even across multiple companies.

3.9.8 Summary and Conclusions

Product managers who have a pure technical background often struggle with the business aspects. However, these business aspects are of utmost importance with

regard to the business success of the product over the strategic timeframe and the product life cycle, respectively. And in most companies the business aspects, in particular the financial model, are the number one driver for management decisions regarding the product. In this section we have looked at business case, business plan, forecasting, cost structure and management, revenue model, and bundling. In addition to these topics, we discuss Pricing in Sect. 3.10 and Performance and Risk Management in Sect. 3.13.

3.10 Pricing

3.10.1 Overview

Software pricing means all activities required to set, communicate, and negotiate prices in a convincing way. The primary objective of pricing is well aligned with product management's objectives, i.e. sustainable success of the software products across their life cycles. In the software business, cost-based pricing works for human services, but not for software products (see Chap. 2). So for software a value-based approach is the way to go. Software pricing means finding ways how to convert the value the software provides to a customer into economic value to the vendor.

This section describes the essence of software pricing. The reader will understand the concept, the related processes, and the impacts of pricing. Software pricing is covered in much more detail in Chap. 5 of [KittClou09].

3.10.2 Concept

Frequently the pricing of their products is not the favorite task of software product managers. If they are lucky they have professional pricing managers by their side. But more often than not, they are on their own. So how does software pricing work? A cost-based approach works nicely for professional services. You calculate the cost per day of your programmers or consultants, add some percentage for general cost and absence, add the profit margin, and voila—that is the price you offer. Unfortunately, this fairly easy approach does not work for software products. There is no significant variable cost, i.e. cost per license or per SaaS customer, that can be used as a basis of the cost-based price calculation (see Sect. 3.9.5). So the total cost per piece goes down significantly with a growing number of pieces, i.e. licenses or SaaS customers (see Fig. 3.7). This is much more drastic for software products than the economies of scale that can be found in manufacturing industries.

So for software, the approach of choice is value-based pricing. That is described by Nagle and Hogan in their Strategic Pricing Pyramid (Fig. 3.8, [NagHog05]):

Strategic software pricing starts with a clear understanding of customer segments and value delivered to the customers (bottom layer of the pyramid).

Fig. 3.7 Cost per piece for software

Fig. 3.8 Strategic pricing pyramid [NagHog05]

Value Creation = The manner in which value is generated in a customer organization from using the product, including the metric that shows the impact of certain parameters on the value.

Segmentation may be needed if value creation is different in different segments. We analyze the value a customer realizes by using the product. Think of an airline reservation system. The value to an airline customer comes with each booking made through the reservation system.

Based on that value analysis, the metrics used in the price structure can be determined (compensator, effect, rating and charging from the model described in Sect. 3.9.6). We define

Price Structure = The manner in which the prices for a given software product are offered, including the metric by which those prices may vary for the single

product (e.g. one single price, price based on number of users, on capacity, on usage, or on volume of licenses acquired).

An important consideration is that metrics should mirror the generation of customer value which gives us a nice story how we can justify the price in relation to that customer value. We may decide to define the price structure as the number of bookings times a base price. Perhaps we want to add a fixed price component to this variable price; after all an airline cannot function without a reservation system and we may not want our entire business being dependent upon the variability in our client's bookings.

There is a wide variety of options available to define a price structure:

- One-time vs. periodic, also known as subscription-based pricing.
- Fixed price (one-time or periodic).
- Usage-based pricing (periodic), e.g. based on number of transactions, users or usage hours.
- Free, i.e. no charges, but revenue generation through advertising.

The resulting price structure may turn out to be quite complex, in particular with usage-based pricing when the actual price is calculated anew every month based on the customer's usage numbers of the previous month. In that case we better make sure that our back-office systems are able to reliably handle the complexity and issue correct invoices. Otherwise we had better simplify the price structure so that we do not impact the customer relationship by frequent incorrect invoices.

Maintenance for license products is usually priced as periodic with an annual charge as a percentage of the list price. The percentage is typically in the range of 12–25%, dependent on what is included in the maintenance contract, e.g. version upgrades, hot line for non-defects, or not. With SaaS, maintenance is not charged separately, but is included in the periodic charges of the SaaS offering.

In some product areas, e.g. airline tickets or rental cars, we see more dynamic pricing concepts where prices change frequently based on current demand, order inventory, etc. This is also known as yield management. These concepts can be applicable to human services. There are examples where yield management is also used for software, e.g. Amazon PaaS pricing.

The better value creation and price structure are aligned, the smoother is the communication of price to customers:

Price and Value Communication = The communication concept that is the basis for communicating price and value to customers by showing how reasonable the price is for the customer compared to the value he gets from using the product.

Strategic pricing also includes processes and policies to ensure the integrity of the price structure in the market, for example fences that prevent abuse of discounts (e.g. student discounts require proof of student status) or criteria for handling "exception requests" in price negotiations (discounting criteria as part of the pricing policy layer in the pyramid). We define

Pricing Policy = A formal definition of the manner in which prices may be altered, e.g. price level or price structure, by whom they may be altered, under what circumstances, and to what degree.

The policy sets governance criteria for the whole company regarding price. If the company has a separate Pricing organization, that organization usually gets a veto right to any transactions that do not adhere to the policy.

The top layer of the Strategic Pricing Pyramid is the price setting, i.e. seeting of the price level. We define

Price Level = The actual amount of charge within the price structure.

In our airline example, for an individual customer, we can calculate an upper limit of the quantified value by dividing the annual profit through the number of bookings which gives us the value per booking. Of course, the price level, i.e. the base price, needs to be considerably lower than the value per booking. Competitive analysis can further help to determine the price level. At what price are competitors offering their products? Does the product have competitive advantages/disadvantages that justify price differences? By carefully analyzing these aspects, we can come up with the price level at which we want to offer the product.

There are many pricing strategies to choose from:

- Premium price strategy (high price, high quality and image), often combined with a promotion price strategy (lower price, high quality, e.g. temporary special offers).
- Price differentiation strategy (same product, different prices, e.g. in different market segments—temporary, geographically, personnel, quantitatively).
- Price bundling (single pricing or product package, see also Sect. 3.9.7).
- Penetration strategy (low price with the introduction of a product in pursuit of rapid market share growth).
- Skimming price strategy (high price for innovative products for compensation of high investment).
- Life-cycle-dependent pricing strategy (situation-specific decision if, e.g. in the introduction phase, high or low prices are to be set).
- Non-linear pricing strategy (usage independent component, e.g. fixed charges, and usage based component, e.g. depending on usage).

The classic pricing theories of economic science like price elasticity usually assume a commodity market and a limited supply of goods. Since these elements don't exist for most software products, usual economic theories do not apply.

3.10.3 Subscription-Based Models

It is hard to run a business based on one-off sales. Recurring revenue makes life so much easier. Once you have acquired the customer, you can be quite sure of a revenue stream over a longer period of time that covers your monthly cost and hopefully more. The insurance industry is built on this model. Employees are used

to it with their monthly salaries. For customers, no upfront investment is needed, and the recurring payments are operational expenses.

The software industry has over time demonstrated the pro's and con's of each approach. In the 1970's and 1980's there used to be a price model called Monthly License Charge (MLC) for mainframe software products. It was the basis of IBM's wonderfully profitable software business. Of course, it means that revenue is deferred compared to a one-time-charge (OTC) model where the customer pays the full license fee immediately when the contract is signed. As a vendor, you need to be in a financial position to afford this deferral. When the PC software business started in the 1980's, the new players like Microsoft wanted the revenue quickly, so they chose the OTC model. They offered maintenance contracts to enterprise customers which generated at least some recurring revenue, but certainly less than the MLC model. It made Microsoft's financial numbers look great quickly. But over time, it meant that Microsoft could only generate additional license revenue from existing customers by charging for version upgrades. So Microsoft released new versions of Windows and Office every 2 or 3 years, not because customers wanted new functionality, but primarily for financial reasons. On top of that, Microsoft often did not care too much about ease of migration. So a large number of their customers resisted that approach and stayed on older versions as long as they could. As of March 2015, only 11.5% of Windows desktop/laptop users were on the latest Windows version 8.1 (according to Net Applications). This does not look good in the trade press, means maintenance efforts for 6 Windows versions in parallel, and results in relatively low upgrade license revenue. With Windows 10, Microsoft was trying something new by offering Windows 10 for free to all Windows 7 and 8 consumer customers in the first year. Did they give away revenue? Well, certainly some. But Windows 10 is the basis for an increasing number of Microsoft applications that come with subscription price models, e.g. the popular game Solitaire will have a Freemium model (use of the basic product is free, but you pay monthly for special features and ad-free use).

The move to subscription price models has become a major trend in the software industry, in particular with Software-as-a-Service (SaaS). It can also be a model for all the other industries that embrace Internet of Things (IoT). Software is becoming a key component of their products. It opens the door to creative value propositions that will make customers pay for subscriptions on top of the OTC price of the base product. Think of software enhancements for entertainment components in the car, or a subscription for continuous high quality traffic updates. So there is lots of room for new ideas how to generate recurring revenue.

3.10.4 Summary and Conclusions

In this section, we have looked at the value-based pricing approach for software and at subscription-based pricing. There are many more aspects to software pricing than we can cover here. For a much more detailed discussion including considerations for corporate IT organizations, we refer to the pricing chapter in [KittClou09].

Pricing, and in particular the governance rules related to pricing, are an ongoing source of conflict. Sales has different, more short-term objectives than a Pricing organization, or whoever is responsible for Pricing. So the governance rules need to define clear responsibilities and escalation paths.

It is imperative to keep in mind that price doesn't sell product. The product must fit the customer need, then price can become the differentiator between doing nothing, buying a competitor's equally useful product, and buying your product. Pricing too low or discounting too deeply, particularly if this becomes predictable, will lead to leaving a lot of "money on the table." Pricing too high will lead to weak market share.

Since pricing has such a significant impact on the product's economic success over its life cycle, the task is always to find the approach which fits the market and optimizes the vendor's financial benefit over time. It can be quite hard to do a convincing customer value analysis. It can also be challenging, at least in B2B markets, to get accurate case by case information on competitors' prices (in some jurisdictions and with some vendor contracts, this may not even be legal to do). And a forecast always contains risk. However, if we want to optimize our business results there are no fundamentally different good alternatives to the value-based pricing approach for software products whether it is an on-premise license product or SaaS.

3.11 Ecosystem Management

3.11.1 Overview

Software ecosystems have a significant impact on the work of the product manager. The product manager must conform to the role the software vendor wishes to play in the ecosystem. The role has direct influence on the positioning of the product, the pricing strategy, the degree to which the product road map and requirements decisions depend on other players in the ecosystem. A product manager can influence the ecosystem strategy and its evolution.

This section describes software ecosystems concept, partner programs and partner management, and the role of software product managers in that context.

3.11.2 Concept

Over the last 20 years the term "software ecosystem" has found increasing usage. It is derived from biological ecosystems. The analogy expresses the interdependence of the players in business networks and was first drawn by Moore in 1993 [Moore93]. We define

Software Ecosystem = A network of people and/or companies that forms around a software vendor or a product or product platform. The relationships in

this network have the goal to achieve benefits for all participants and can be formalized or not.

The structural constituents of software ecosystems are stakeholders, relationships, boundaries, behavior, and strategies. Iansiti and Levien [IanLev04] define three principle roles participants in a business ecosystem can play:

- Keystone:
  A benevolent hub in the network that provides benefits to the ecosystem and its members. It usually provides the core of the innovation in an ecosystem. For technology-related ecosystems, this effect of keystone players is a key success factor for survival and adaptability of the overall ecosystem. Keystone players behave in favour of other players, especially by protecting niche players. The number and diversity of niche players determine the speed and diversity of innovation in an ecosystem, which is an important prerequisite for success. The resulting distributed nature of the network makes it flexible and little vulnerable to external disruptions. Examples are Google in the Android ecosystem or Microsoft in the Windows ecosystem.
- Dominator:
  A hub that aims at controlling as much space in the network as possible. It leverages a critical position in the ecosystem to exploit or take over a large portion of the ecosystem. It is not interested in sharing value, but tries to capture most of the value itself. An example for a dominator is Apple who controls the IOS ecosystem to a large degree.
- Niche players:
  Most members of an ecosystem are niche players who do not try to compete with a keystone or dominator, but focus their business on critical competencies in narrow areas of expertise, if there is an opportunity to run a profitable business. They usually are smaller companies and outnumber keystone players or dominators. An example for niche players are app developers in the ecosystems of smartphone platforms like Android or Apple's IOS.

It is usually not part of the responsibilities of a software product manager to decide on the role his company wants to play in an ecosystem. This belongs into the realm of corporate strategy that is owned by executive management. Part of this decision is how proactively a company wants to influence the ecosystem and other players in it. Once these decisions are made, they have significant impact on the work of the product manager, and he must conform to the role his company wishes to play in its ecosystem(s).

Among the niche players, there are different types of players in an ecosystem:

Development-related:

- **Independent Software Vendor (ISV)**: Develops functional extensions, i.e. add-on or specific application as his product that uses the platform product as a prerequisite and enhances its functionality. He is independent in the sense that he is not controlled or owned by another vendor.

- **Original Equipment Manufacturer (OEM)**: One company integrates the product of another company into its product offering.
- **Technology provider**: Provides relevant products and/or technology with interfaces to the platform product.
- **Resource Provider**: Provides code that is integrated in other players' products, or sends developers into other players' development teams, or provides operations services to other players.

Sales-related:

- **Reseller**: sells software products offered by a software vendor. The reseller has contractual relationships with the customers and with the vendor.
- **Value Added Reseller/Remarketer (VAR)**: a reseller that offers a software product offered by a software vendor and adds components and/or services to them, e.g. customer-specific customizing. The VAR is helpful for better market penetration, and sometimes for enrichment of products with solution components.
- **Intermediary**: mediates between vendors and customers of software products, with the aim of them signing a contract, e.g. a provider of an internet marketplace.
- **Original Equipment Manufacturer (OEM)**: The development relationship described above can also be considered as a type of sales and distribution outsourcing. The product integration can be visible to customers (black label approach) or invisible (white label approach).
- **Value added distributor (VAD)**: is used for outsourcing of production and distribution activities, often for enrichment of products with solution components, for management of smaller partners and a better market penetration.
- **Technological alliance**: is often used as sales cooperation, for preinstallation, for completeness of solution offers and synergy in marketing.

Service-related (services to potential customers of products in the ecosystem):

- **Consultants**: Company or individual who offers solution definition (incl. product selection), business analysis etc.
- **System integrator (SI)**: coordinates and performs the integration of software product components supplied by different vendors and customer-specific software. This includes the responsibility for the overall system design as well as the customizing and integration of product and service components and information management.
- **Operations Provider**: offers managed services like hosting, maintenance, etc.

Influencers:

- **Consultants**: may influence customers' buying decisions through their consulting work.
- **Press**: may influence customers' buying decisions through their coverage of relevant markets, products and companies.
- **Market research companies**: may influence customers' buying decisions through their coverage of relevant markets, products and companies.

- **Customers**: may influence other customers' buying decisions through their testimonials and their acting as reference customers for vendors.

An individual company can often be a player of more than one of these types. Competitors to the platform product or technology vendor of an ecosystem are usually not considered as part of the ecosystem unless there is a coopetition relationship, i.e. the same companies cooperate in one or more product areas and compete in other product areas.

If a company decides to influence its ecosystem and other players in it proactively it usually establishes some kind of partner program. We define

Partnership = Formalized relationship in a software ecosystem.

A partner program can be applied to all types of players in an ecosystem except influencers. For software more than for other products, a well-oiled network machine of diverse partners is a significant prerequisite for long-term success in addition to the traditional direct sales channels. For a software company, there are several good reasons for establishing and managing a partner network as part of a software ecosystem. Partners are needed in order to reach customers that could not be reached fast enough or at all with a direct sales force. To build up a sales organization is time and cost-intensive and requires significant revenue per person. A sales partner may be able to cover a sales area more easily and efficiently by combining products of several software companies and his own products and services. Moreover, many sales partners are already established in the market with customer contacts as value added resellers (VARs) or system integrators [Bech15].

Customers of business software are usually not interested in a particular software product, but in a solution for their business problems and an implementation of their business processes as efficiently as possible. Because of the complexity and interdependence of business software, the desired solution is typically not achieved by a single software product and requires significant customizing and implementation effort. Customers frequently rely on qualified consulting and service companies that are contracted for the implementation and very often also for conception and design of the solution. This gives these companies a strong influence on the selection of software products, sometimes the decision is part of the contracted task. That is why partnering with these consulting companies can be more important for a software vendor than direct sales. This is valid for application software and especially for infrastructure software products. In many cases, the decision for a certain software technology or a software product is not made by the customer, but by the vendor of an application software solution who decides for a base technology, or by the system integrator who favors a technology or product and uses it as an integral part of his offer.

Last but not least, vendors whose products are not competitive, but cooperative, form technology alliances. These alliances have the purpose to bundle sales resources. This can happen as sales cooperation, or by preinstalling software products on the hardware platform of another manufacturer. The advantage of preinstallation for the hardware manufacturer is that the additional software

products make his product more attractive. The advantage for the software vendor is that this channel allows easy access to many end users that would otherwise fail to be addressed in the usual sales process. In view of the high sales volume, the purchase commitment, the negligible amount of time and effort invested in sales and distribution, and perhaps even the money saved on media and documentation, the software vendor will offer to sell his software on very favorable conditions. Technology alliances also aim at offering integrated solutions that the market requires by combining products and services of several partners so that the customer can assume a proven level of integration. Often partners want to benefit from the image, the brand or the market position of other partners.

In the end all partnering activities have one goal: faster growth and market leadership. A balance between growth and profitability needs to be found: The software vendor gives some part of the revenue and/or profit to its partners in order to grow faster and reach a leading position in the market.

A comprehensive discussion of business ecosystems can be found in [JanBrCus13]. Adnar and Kapoor [AdnKap10] look at what technology leadership and innovation mean in an ecosystem.

3.11.3 Partner Programs

Depending on the role that a company wants to play in a particular ecosystem, the company needs to be visible as an active member of the ecosystem and influence and support it. This work is typically split between Product Management, Development and Marketing. When larger partner programs generate a significant workload and require a uniform management of terms and conditions, a centralized partner management organization may be the appropriate organizational solution. But even then, product management needs to be involved on a strategic level. Software product managers are well advised to participate in, contribute to and try to influence the ecosystems important for their products. Partnerships and alliances that are required to actually provide a complete offering to the customer need special attention.

A partner program typically specifies:

- Structure of the partner program, e.g. different types and levels.
- The prerequisites that a partner must fulfill for participation.
- Contributions of the software vendor, e.g. for joint marketing, education, early technical information in case of changes, support, or platform for information exchange among members of the program.
- Terms and conditions, in particular with regard to sales.
- Additional rules, e.g. for joint marketing and sales activities.

When selecting partners, the strength and stability of a company, compatibility of business models, potential conflicts, market access, skills, and the level of commitment are important criteria (see also [FoFrFiLG14]). Partnerships only make sense

when they are beneficial to both parties and the parties trust each other. This win–win-character needs to be ensured when designing terms and conditions of partner programs or negotiating individual partnerships.

SAP is an example for a company that has an exceptionally strong focus on partner management with a partner management unit reporting directly to the CEO. Marketing is responsible for sales-oriented partner management (see Sect. 6.3). Here Product Management is involved in dealing with the partner's product requirements. Product Management takes care of product-related in-depth discussions with market research companies and journalists, the overall positioning within the ecosystem, and the selection of product-specific partners.

3.11.4 Summary and Conclusions

Software ecosystems have a significant impact on the business success of its players. In this section we have looked at the different roles and types of players. Companies can intensify cooperation within an ecosystem by establishing a partner program or joining other players' partner programs. While the big software companies invest a lot of resources in extensive partner programs in order to manage their ecosystems proactively, smaller players often do not give sufficient attention to this subject. In any case it is part of the software product manager's responsibility for product strategy to influence his company's approach to ecosystem management for the benfit of his product.

3.12 Legal Aspects

3.12.1 Overview

Software product managers are usually not legal experts. And they do not have to be. However, since legal risks can have a significant negative impact on the business success of a product, product managers need to be aware of those risks and take action to avoid or mitigate them. It is advisable to involve legal experts in these topics. But a product manager needs to know the important questions to ask the legal experts. That is what this section is about. We cover contracts, the protection of intellectual property, open source, and data protection in more detail.

3.12.2 Contracts

In this section the focus is on contracts between vendor and customer. The chosen delivery model, i.e. licensed product vs. Software-as-a-Service (SaaS) offering (see Sect. 3.5.2) determines the type of contract. A license describes to which extent and under which conditions the licensee can use an item, e.g. software, which is subject to intellectual property rights, in particular trademarks, patents or copyrights. These

rights of use are granted by the licensor who may be the owner of all rights in such item, i.e. the software company, or may be an entity authorized to grant the license, e.g. a reseller. The term "license contract" means an agreement between the licensor and the licensee about all terms in connection with one or more licenses. With software, the term "license" usually describes the scope of rights of use which the licensor grants to the licensee. In case of SaaS, the customer does not acquire a license, but a service.

The contract by which software or a software-based service is "acquired", may be negotiated individually or, particularly in the mass-market, based on so-called "standard terms and conditions," which describe the generally applicable legal terms.

The **scope of the license or service** describes the content of the agreement, i.e. the scope of software, documentation, services and hardware included. Vendor services (e.g. for custom modifications, initial migration work and for user training) and further services such as additional copies of the documentation are usually defined in a separate services agreement. It is worthwhile here to explain an accounting principle which will be important to the vendor (and which may explain certain vendor behaviors to the customer): in most countries, revenue for OTC software is generally bookable upon delivery of usable product and invoice to the customer. However, if there are services to be performed or an acceptance test following modifications or there is a right of the customer to cancel, then the revenue will not be bookable until all of the contingency conditions have been performed and are accepted. In today's frenetic scramble for more and earlier revenue, such contingencies will be very hard for a vendor to accept.

One of the peculiarities of software is that inherent in most contracts there is an assumption by both the vendor and the customer that the software contains errors (bugs). That is why the terms and conditions of the license must define how such bugs and any necessary modifications are to be dealt with. That is also why there is **no guarantee** with software except possibly a money-back provision if the customer is not satisfied; this generally takes the form of a trial period after which you must pay or you will no longer be able to use some or all of the functionality or a "pay now and we will refund your money within xx days if you are not satisfied". The "xx" usually ranges between 30 and 90, seldom longer. Both of these approaches, while helpful to the customer raise revenue recognition issues for the vendor.

Warranty means that the vendor agrees to make the product perform as promised in specifications, advertising, or sales presentations during a limited time. The warranty period is typically 1 year, in some jurisdictions 2 years, unless specifically stipulated otherwise. Assistance with debugging and provision of fixes are usually free of charge during the warranty period, i.e. included in the license price. This situation may continue longer if maintenance is paid for the license. In case of SaaS, maintenance is part of the scope of service anyway.

In this area the definition of a "bug" is always a critical item. A frequent matter of customer/vendor dispute is determination of the severity of a bug and whether or not it really is a bug or rather a request for product enhancement. IBM says that if

the code does not do what the product specification says it does, that is a bug. However, on occasion, when a bug cannot be fixed, the specification is changed rather than the code. It may be impossible to achieve a legal definition that is absolutely airtight.

This part stipulates whether or not the licensee may transfer the license to another physical computer, how to proceed within a local area network, and how copies of the code are to be handled. Another critical issue is whether or not the licensee may **transfer the license** to a third party. In the European Union, there is a High Court decision that this kind of transfer is generally allowed. In other jurisdictions, the vendor may still try to prevent this.

Under **type of charges** there is not necessarily the specific price, but possibly how the product is charged in terms of metrics. This is a candidate for a product-specific addendum to a generic vendor license.

Vendors try to limit their **liability** as much as possible.

Usually if a customer wants to continue to use the debugging service described under "Warranty" and have access to fixes past the end of the warranty period, he must acquire a maintenance contract. We recommend vendors to keep the **maintenance** contract separate from the license contract. In the maintenance contract, the vendor will define the entitlements being conveyed, charging and duration, and termination provisions. The vendor may also define a support escalation process which brings more resource to bear depending on the severity of the bugs. But see also under "Warranty" what a fix may entail. Some vendors also include upgrades to new product versions in their maintenance agreements. Various other aspects of the maintenance services may also differ from vendor to vendor.

In general, a vendor reserves the right to terminate maintenance for a software product or product version, the common practice being to give notice of this in advance. The vendor does this with the aim of reducing maintenance costs and motivating customers still using older product versions to migrate to new ones. The announcement that maintenance services are to be discontinued is usually not received very well by the customers involved. This regularly gives rise to fierce protests that are then settled by reaching a compromise.

Different geographies will have different views on this. Different product sets may attract different reactions. In Japan there is a tendency to stick with a product for a very long time, to the extent of being willing to pay extra for product "n" maintenance in order not to have to move to "n + 1". This is an extension of the adage "if it ain't broke, don't fix it." But this can also delay introduction of new technology. Microsoft recently experienced this when they announced the discontinuation of maintenance for Windows XP.

Then there are miscellaneous legal provisions that include the definition of the period during which one can make claims, governing law, and which courts will be used in case of dispute. Other legal provisions to be included pertain to the mutual handling of confidential information as well as to the rights and responsibilities entailed in terminating the license agreement, legal remedies and issues of compensation for damages and liability. A corporate customer in doubt about a producer's soundness may insist on a source code escrow (escrow service). This

allows the corporate customer to access the source code in the event that the producer declares bankruptcy, thereby ensuring that maintenance can be continued. While customers sometimes ask for this, it is usually more a bargaining chip in negotiation than a practical request. The source code of an operating system, for example, would not be helpful for a customer. Some vendors like IBM even have large chunks of code in proprietary languages for which compilers are not commercially available. And where would the customer find the resource and talent to work on such code? Often customers lose interest in such a request when confronted by the cost of having an escrow agent keep a copy of the code.

Conflicts between vendor and enterprise customer generally arise in one of four areas:

- Whether the product is doing what the vendor claimed it would do and if not whose fault it is that it is not performing;
- The way in which the product is being used compared to what the vendor authorized; and
- The quantity of product in use compared to what the vendor authorized (for license contracts);
- Unused quantities of the product which the customer paid for but which he is not using, which is referred to as "shelfware" (for license contracts).

The license contract attempts to deal with most of these items. Examining the list of potential conflicts between vendor and customer, one can conclude that most of them arise because a software "purchase" typically does not mean that ownership or any proprietary rights are transferred as they usually are in the case of material products. Only a right to use is granted by the vendor, subject to various restrictions and often priced based on the scope of use or projected use. This circumstance gives rise to a legal complexity that results in similarly complex licensing terms and conditions, the details of which frequently differ from one country to the next, since the local legal requirements must always be taken into account.

Contrast the acquisition of software with that of a tangible widget: if you buy a shovel, you can use it 1 day to dig a trench, lend it to a friend the next day to mix cement, and your wife may use it to plant roses the third day. In fact, if one discovered that he had purchased too many shovels, he could try to sell his extras. Not so with software which will probably have restrictions on the purpose for which it may be used, the number and nature of the people who may use it, and possibly on where and how it may be used and/or resold. The wording of the terms and conditions of the license and related documents has by now become so creative that in some states in the USA, e.g. California, initiatives have (unsuccessfully) been undertaken calling for a law to regulate standard software license agreements, as the terms and conditions of license agreements now used by vendors can often no longer be understood by either retail or corporate customers nor on occasion by the vendors themselves.

In the case of **SaaS** or other software-intensive services, the service provider, i.e. the entity which makes available the software through the internet, needs to

either own the IP rights in the relevant software itself or has to conclude license contracts with the owner of the software which explicitly allow this kind of use, in other words a SaaS-license. Customers of such a service do not need license contracts, but only service contracts with the service provider, as the object code is not running on their system, but only on the service provider's infrastructure (whether owned or leased). Please note that this applies to SaaS-scenarios. In IaaS-scenarios the license requirements may be different: the IaaS-provider or the customer may need a license for the software installed that runs on the infrastructure provided by the IaaS-provider.

SaaS contracts usually contain a **service level agreement (SLA)**. SLAs stipulate which services a customer can expect to receive from a provider. In particular, this may include:

- Functional scope,
- Availability as an average as well as the maximum length of downtime,
- Quality in terms of absolute errors per time period according to degree of severity and maximum time for debugging according to degree of severity,
- Performance guarantee, e.g. response time behavior or load values,
- Scope and quality of user support, e.g. hotline, and maximum reaction time to operational demands, e.g. special evaluation of data on hand,
- Reaction time to new functional demands,
- Test options for new releases and versions including the scope of support and maximum debugging time according to degree of severity,
- Backup frequency and maximum restore time,
- Disaster recovery measures.

From a customer perspective, an SLA only makes sense if the contract includes penalties that must be paid by the service provider in the event of a breach of the terms defined in the SLA. The purpose of this is to motivate the provider and compensate for damage incurred by the customer in the event of a system failure.

A corporate IT organization acts as a service provider for the company's other business units. The services provided include further development and operation of the software and hardware used to support the business processes of the other departments. Whereas in the past the relationship between the in-house IT department and other departments was often informal, it has become common practice to use formal service level agreements (SLAs) to define this relationship, especially if the IT organization has been restructured as a legal self-contained corporate entity, almost like an outsourcer.

3.12.3 Protection of Intellectual Property

Software product managers normally deal with product evolution. The task of bringing products to market that are based on real groundbreaking innovation will be the exception. In such a case, the software product manager is often not involved

until the innovation has reached a stage at which it can be included in the normal development process. Regardless of whether it is a question of evolution or innovation, the results can lead to a significant market differentiation and to major competitive advantages. The software product manager will then be responsible for finding a way to turn these competitive advantages into market success and safeguarding the competitive edge as long as possible.

There are a number of different legal constructs for the protection of intellectual property. From a macro-economic view, protection leads to more innovation since it increases the return on investments in innovation. On the downside the use of the innovative idea by others is delayed or made more expensive, since it cannot be directly copied. An example is a pharmaceutical company that develops a new drug and keeps the price high, thereby limiting its use.

On the micro-economic level, if a company seeks legal protection it needs to make a description of what it wants to protect public which makes it easier for competitors to understand it and duplicate it legally (and sometimes illegally). Since this can be difficult to prove with software, smaller companies sometimes avoid publishing their innovations, whereas larger corporations mostly prefer it. The advantages are the protection of competitive advantage, possibly an extra income through license fees, negotiating leverage in case of conflicts with other patent-holding companies, and the marketing aspect of an innovative corporate image.

Of course, there is no automatic correlation between the innovative capability of a company and the number of patents it holds, since the acceptance of a patent does not signify that the content is useful. So far, the general public does not seem to have realized this.

There are four fundamental legal constructs for the protection of intellectual property:

- **Trademark**: Protection for the names of brands, i.e. brands do not apply to the software itself but only to the brand under which it is marketed.
- **Trade Secret**: Protection of company-internal knowledge (primarily against employees). This protection is exercised by restricting knowledge and access to a very small number of people and by using non-disclosure agreements. In most jurisdictions, trade secrets are only protected under unfair competition laws.
- **Copyright**: Protection against copying of software code (as specific expressions of an idea or way of doing something) and product material such as manuals, brochures and presentations. This is the main way software is protected. The algorithm or idea behind software is not protected under copyright law.
- **Patent**: Protection of the specific technical concept or idea. In most jurisdictions, patent protection can only be obtained for software which is integrated into a technical solution to a problem.

Traditionally, text has been subject to copyright law, technology to patent law. Since software became an issue, legal authorities and software developers, seeking

protection for their intellectual property, have sought to choose existing recognized forms of protection under the law and have tried to press software into these existing alternatives (see [Klemens06]) not accepting the fact that it does not really fit. This has led to continuous controversial discussions and country-specific regulations that are not aligned internationally.

In the **US** up until 1980, patent rights were hardware-oriented, i.e. it was hardly possible to have software patented. In response to pressure from software vendors, a Supreme Court decision changed the rules. Suddenly almost everything was patentable, even business processes. This resulted in a glut of patent applications, some of which seem rather useless or even ridiculous (see [Klemens06], p. 1 ff.).

As long as software patents refer to implementation details, they can be circumvented by implementing the functionality in some other way. This becomes more difficult if user interface elements that implicitly describe a business process are patented (see the Amazon example below).

Some years ago, conflicts arose regarding open source software and patents, as companies introduced patented elements into open source processes and subsequently requested license fees. All open source groups refused to accept this, arguing that using patents this way prevents progress. In 2002, open source guru Richard Stallman said that U.S. patent logic would have forced Beethoven to pay Mozart for the right to create a new symphony.

The monetary judgments that have been sought and sometimes granted in patent infringement cases in the US are increasingly seen as life-threatening for participants in the US software market.

The way that patent offices review software patents is also being criticized more and more. Officially, an application may not be accepted as a patent if its content already constitutes public property, i.e., whatever is already generally being used may not be patented. Unfortunately, there are good examples showing that patent offices were not capable of judging this. In [Besaha03] Besaha proposes measures for improving the patent process. In two ground-breaking rulings in April 2007, the US Supreme Court decided to lower the requirements regarding the non-obviousness test (Teleflex vs. KSR International) and restricted the applicability of US patent law to US territory (AT&T vs. Microsoft). The America Invents Act of 2011 brought some improvements, but no fundamental change.

Amazon is a widely discussed example of how patents can be used to safeguard the competitive edge of a product platform. Amazon had its one-click technology patented, making the ordering process extremely easy for registered customers. This discussion revolves around the issue of patentability. Opponents argue that, ever since the invention of the mouse, a click has always been used to effect a transaction. In their opinion, this technology is therefore not patentable. Proponents say that this special type of one-click technology used in connection with the online commercial transaction process for which Amazon has submitted a patent application did not previously exist in this form and can therefore be patented. In Alice Corp. v. CLS Bank International, the US Supreme Court decided in 2014 that an abstract idea cannot be patented just because it is implemented on a computer. Since then, hardly any patents on pure software or business processes have been granted in the US anymore.

This subject is being heatedly discussed in **Europe**, too. In compliance with the current European patent agreements, neither a computer program nor a business method can be patented as such. However, since it is not clear how a computer program differs from an invention that includes a computer program as one component among others, the European patent office has accepted software-related patents in the past few years. The urgency for a European patent law is obvious since the heterogeneous legal situation within the European Union means costs for patent applications are higher than in the US by a factor of 10. The European Commission has repeatedly started legislative processes in order to establish more explicit common regulations. The latest one is the European Unitary Patent accepted by the European parliament in 2012. It is unclear how the required ratifications by the European countries can procede after the UK's Brexit decision.

Changes in the legal situation in both the USA and Europe regarding the granting of software patents can be expected on an ongoing basis given the discussions described above. Yet, regardless of how the legal situation develops, the software product manager will still be responsible for considering a patent application as a means of protecting intellectual property and safeguarding competitive advantages.

There are additional ways and means to do this. A key factor is assuring that valuable technical employees continue to be committed to the company. Studies show that typically $<10\%$ of the development staff actually possess the essential product- or technology-related skills. These are the employees that the company definitely wants to retain. This requires company-wide personnel retention programs that are generally not within the software product manager's scope of responsibility or authority. However, the software product manager needs to ensure that the employees relevant to his product are included in these programs. The competitive advantage can also be maintained by continuing to expand the differentiating elements through further product development.

A major problem for all vendors selling software licenses is software piracy. IDC conducts studies for the Business Software Alliance (BSA) on global software piracy quite frequently. According to the latest study [BSA16] 39% of all software installations worldwide are not properly licensed leading to a total revenue loss of 52 billion & in 2015. The piracy rate ranges from 88% in Venezuela to 17% in the US. Even if these numbers are a bit inflated there is no doubt that there is a significant problem and only slow progress. Many large software vendors now insist on the contractual right to audit customers as one way of curbing losses. That of course offers no protection for software usable by single clients, pirated versions of which are often available in Russia, China, and elsewhere for a fraction of the normal cost. Microsoft now checks routinely whether the requestor has a legitimate license before allowing downloads of maintenance and upgrades. There are quite a number of publications on what companies can do to prevent piracy, e.g. [CollThom02] and [GopSand97]. One of the attractive features of SaaS for vendors is that it eliminates the piracy risk.

3.12.4 Open Source

On the sourcing side (see Sect. 3.8), Open Source software is frequently an (important) part of software development projects and may help to reduce development cost and time (see [Popp15]). However, as all software, open source software is subject to copyright protection. Thus, the developer of an open source software component is owner of the pertaining copyrights and disposes of them by offering the module free of charge under an open source license agreement. Consequently, a company which uses open source software for its own development processes has to comply with its license terms. An according license contract is frequently concluded (implicitly) when downloading or installing the open source component.

The variety of available open source licenses is manifold [DeLaat05]. From the perspective of the user of open source, the so-called free licenses are not causing any problems. These licenses simply allow any kind of use of the open source software free of charge without stipulating any further restrictions (e.g. Creative Commons' CC0, BSD License, MIT License).

More difficulties can be caused by the so-called copyleft licenses which oftentimes govern the use of open source components [e.g. the GNU-Licenses (GPL, LGPL and GFDL)]. The underlying idea of copyleft licenses is that the open source software is made available free of charge and thus any further developments accomplished on the basis of such copyleft software may only be distributed on the basis of the copyleft license. In practice, this means that the source code of software which was developed by using open source software must be offered free of charge—at least to anyone who acquires a copy of the object code, sometimes also to the general public. Dangerously, it does not matter how significant the open source code was in developing software. If only an open source code snippet is used, the respective copyleft license applicable to such snippet applies to the entire software (so-called viral effect). As long as such software is used for mere internal purposes, the copyleft license effect does not apply. As soon as the object code is sold (stand-alone or as part of a hardware product), the obligations of the copyleft license apply, in particular the above mentioned duty to make available the source code, and formal requirements, such as the obligation to mention the applicable open source license in the source code.

If open source code from various origins has been used, multiple open source licenses may apply in parallel—making it impossible to comply with all of them (in case of distribution). In case of an infringement, the owner of the rights (as a rule, the developer(s) of the respective open source code) have the right to request halting the use of the software and to claim damages. Damages may e.g. be calculated by the amount of profit made by distributing the software in violation of the open source license. Ruffin and Ebert [RufEbe04] look at the risks in more detail.

Companies who are developing software should implement open source compliance processes. In this process, a list of open source licenses which are compliant with the intended distribution model is set up and developers (employees, freelancers and subcontractors) are prohibited from using any open source software

for which other than the listed licenses apply. If a developer requests the use of non-listed software, the pertaining licenses have to be checked by the legal department (or outside counsel). In case the respective open source license, e.g. copyleft, is not in compliance with the intended distribution model, the open source code should not be made part of the proprietary software to be developed. An option to nevertheless use the respective open source software, is to make it a part of a separate component which interacts through defined interfaces with the rest of the software. This approach may allow the software-developing company to avoid the obligation to make available the source code of the entire software free of charge but limit such software to the component comprising of open source software.

A disputed topic is whether or not SaaS has to be considered as distribution under the copyleft licenses. Only few open source licenses provide for specific and clear rules in this respect. In lack of clear license clauses, the implications are unclear. It can be argued that SaaS is not distribution since the object code is not sold and not transferred to the customer. However, the purpose of copyleft licenses is to ensure free availability of the source code in case of commercial use. The Free Software Foundation, issuer of the most common copyleft license family, the General Public License (GPL), has expressed the view that distribution via SaaS does not equal distribution in the sense of the license. However, there are a number of open source licenses now, e.g. AGPL, that explicitly address the SaaS scenario.

3.12.5 Data Protection

European Union (EU) data protection law applies to all companies and branch offices within the European Economic Area (EEA) and also to other companies to the extent they collect, process or use personal data with means located in the EEA.

The EU data protection law has served as a model for many other jurisdictions. Consequently, various other countries have enacted data protection laws which are meeting the EU data protection requirements, or are even more restrictive: e.g. Switzerland, Canada, Israel, Argentina, Australia etc. However, in many other countries, e.g. in the USA, the approach to data protection is fundamentally different. Whereas in the EU any kind of personal data (i.e. any data which relates or can be related to an individual) is protected irrespective of its sensitivity, in the US numerous special data protection laws apply in certain areas, e.g. for health data or credit card data; outside those areas the general principle of privacy applies which only grants protection to sensitive data from private spheres. As a result, e.g. personal data of employees is subject to the data protection laws in the EU, but not, at least in principle, in the US.

Under EU data protection law the company running software to process personal data, e.g. ERP software, CRM software etc., is responsible for data protection compliance. Hence, the software-developing company is not directly in charge of data protection compliance. However, customers are, and will become more and more, aware of data protection issues and will thus likely request software which takes into account data protection principles and requirements, such as data

minimization including pseudonymization wherever possible, access by users on a need to know basis only (roles) and data security. Moreover, the new EU data protection regulation that came into force in May 2016 establishes the principle of privacy by design and the right to be forgotten, among other regulations. European countries are supposed to translate it into national law by May 2018. It will bind software companies to design their software in a way reducing the risk of data protection infringements by the users. It remains to be seen to which extent data protection authorities will address software companies directly in the future.

The situation is fundamentally different if the software company runs the software and makes it available to its clients under a SaaS distribution model. In such case, the client's personal data are processed on the company's server infrastructure. Nevertheless, the customer remains the controller of its data who remains responsible to ensure data protection compliance. As a rule, the SaaS provider is considered to be a commissioned data processor. In such case, the controller and the SaaS provider have to conclude a commissioned data processing agreement according to which the SaaS provider commits to comply with the directions of the controller, allows regular checks and controls and implements adequate technical and organizational measures for data security. Special restrictions may apply in certain areas, e.g. telecommunication, health data, insurance data, tax data etc.

To the extent that personal data is transferred from the EEA to recipients outside of the EEA, SaaS requires additional precautions if the recipient's country has not been approved by the EU commission as a safe country. The typical solution is that the controller and the SaaS provider agree on the so-called EU model clauses which oblige the data importer (i.e. the SaaS provider) to comply with the fundamental principles of EU data protection law. For recipients in the USA, the safe harbor rules applied till they were declared as invalid by the European Court of Justice in October 2015. Its replacement, the EU-U.S. Privacy Shield, came into force in July 2016. If a US company registers under the EU-U.S. Privacy Shield and commits to comply with its rules, it is considered to ensure an adequate level of data protection.

The advantage of commissioned data processing is that the transmission of personal data from the customer to the SaaS provider (and back), which is triggered by the use of the software, is as a matter of principle not considered a transfer in the sense of the law. However, as mentioned above, this applies only provided that the SaaS provider has no discretion how to process the data and does only process the data for the customer's purposes. If this precondition is not fulfilled each and every transmission of data between the customer and the SaaS provider needs to be based on the data subject's consent or a statutory permission.

3.12.6 Summary and Conclusions

Software product managers need to be aware of the legal risks related to their products which can potentially harm the business success of the product significantly. And they need to take action to address these risks in cooperation with legal experts. We have described risks and in the areas of contracts, protection of

intellectual property, open source, and data protection. There are additional risks such as governance, finance, supply-chain, delivery commitments, laws on general terms and conditions, blacklisting of countries for specific software components etc. which are increasingly relevant (see for instance the growing impact of governance rules and transparency laws).

The term "compliance" is widely used now describe the goal that organisations aspire to achieve in their efforts to ensure that they are aware of and take steps to comply with all relevant laws and regulations. Software vendors can achieve competitive advantages in B2B markets by helping their customers to achieve compliance.

3.13 Performance and Risk Management

3.13.1 Overview

Once the product strategy is defined business measures or key performance indicators (KPIs) are needed for continuous tracking and analysis of the business performance of the product. Ideally, all elements of the product strategy are addressed. The measurement results help the organization to learn and improve, and to track whether the product is following or drifting away from the business model and targets and take appropriate action.

As with any business activity, there are inherent risks that need to be identified and managed. Risk Management means the continuous tracking and analysis of risks identified in connection with the software product be it in development, sales, customer use or anything else. Again appropriate and timely action needs to be taken if needed.

3.13.2 Performance Management

Figure 3.9 shows examples for frequently used measures in four perspectives. Often the measures are standardized on the corporate level. If, however, the software product manager has the freedom to define them a trade-off needs to be determined what ought to be measured and what can be measured in a relatively simple and cost-effective manner (see also Sect. 5.6).

An example for a comprehensive measurement approach is the Balanced Score Card (BSC) [KapNor96a, KapNor96b]. Balanced Score Card (BSC) is a suitable framework for translating a product's strategy into a coherent set of performance measures. It complements traditional financial indicators with measures of performance for customers, internal business and innovation and improvement activities. Although a comprehensive framework, BSC needs to be carefully planned and implemented as it is complex, time consuming and expensive to define and track relevant measures.

Perspective	Business canvas	Sample measures
Financial	Revenue streams and cost structure	Profitability, revenue, market share, acquisition costs
Customer	Value proposition, customer relationships, customer segments	Product use, customer satisfaction, customer perceived value, retention rate, number of registered users, number of active customers, number of installed licenses, conversion rates
Internal Business	Key activities, key partners, channels	Process quality, process cycle time, product quality, development productivity
Innovation and Learning	Key resources	Human capital measures

Fig. 3.9 Mapping of the Balanced Score Card's four perspectives to business canvas segments and sample measures

BSC's financial perspective (see Fig. 3.9) defines the long-term objectives of a product. While profitability is the most emphasized financial objective, other financial objectives are also possible depending on the life cycle stage of the product. For example in the growth stage, sample measures are sales growth rate by segment, percentage of revenue from new product, in the maturity stage, sample measures are cross-selling, indirect profitability, cash-to-cash cycle, in the decline stage, sample measures are percentage of unprofitable customers etc.

In the customer perspective of the balanced score card, product managers identify the customer and market segments in which the product will compete and the measures of the product's performance in these targeted segments. Examples of such measures include customer satisfaction, customer retention, and new customers' acquisition.

In the internal business perspective of BSC, managers identify the critical internal processes in which the company needs to excel in order to deliver the product successfully. Sample measures include process quality, process cycle time.

The innovation and learning perspective identifies the infrastructure needed to create long-term growth and improvement. Sample measures include human capital measures (employee retention, employee satisfaction, employee skill, structural capital etc.).

Using all four perspectives together helps to reveal the existing gaps between people, systems and procedures that need to be bridged for delivering the promised value to customers and meeting company's financial objectives. It is important to note though that the measures related to financial and customer perspectives are usually owned by the respective product managers while the respective process owner usually owns measures related to the internal business process and innovation and leaning perspectives. However, the product manager can use these measures for performance measurement of the product.

The choice of measures has to be a careful one since the chosen metrics should enable effective decision support and be simple to measure (see also Sect. 5.6). [FotrFric16] shows how analytics can help product managers with the decision-making. While so-called vanity metrics make a company feel good, they do not

provide actionable insight. Thus, a product manager needs to select actionable metrics (that provide insight and support decision-making). For example, during the growth phase of a paid web service, it is recommended to continually increase the budget for customer acquisition initiatives to take advantage of the growth opportunity. However, in that situation, focusing solely on the number of registered users in most cases is a vanity metric—what really matters in that situation is the number of users relative to effort/acquisition costs. If the number of users is increasing by 50%, it might look good at first sight (vanity metric), but if the average acquisition cost per user shows a trend of increasing heavily over time, it is actually not a good development. There is a risk that the product will run into a situation where acquisition costs per user are getting higher than the average life time value of a new user. That is clearly not sustainable—every new customer will increase the company's losses. In this example, the standalone "number of users" metric is not really actionable for product managers, as it is not telling them whether their product is moving in the right direction or not. Here are some examples for actionable metrics:

- A/B tests: a way of comparing multiple variants of a product to find out which one works best in terms of predefined objectives and measurements.
- Per-customer metrics: customer-centric measurements.
- Funnel metrics: measurements related to marketing and sales success.
- Cohort analysis: measurements focused on subsets of customers with common characteristics.
- Keyword metrics: measurements focused on search engine optimization.

With software, product usage can be measured in ways that no other product area allows. For a web-based environment the Lean Startup movement provides a lot of information on how to track users and develop meaningful metrics (see [CrolYosk13]). This data allows highly valuable insight into how individual product features are being used or not used.

Performance Management is tightly linked to Product Analysis (see Sect. 5.6) and Product Life Cycle Management (see Sect. 4.4), and Planning Processes and Tools (see Sect. 4.5). With Product Analysis, the organization ensures that the relevant product-related data becomes available in a reliable and frequent way. Market Analysis (see Sect. 5.5) does the same for market-related data. In Product Life Cycle Management, Product Management uses this product and market data to determine the positions of the product in its life cycle and of the market (product category) in its life cycle. Depending on these positions, different measures will be prioritized for Performance Management (see Sect. 5.6.3), in which Product Management tracks the corresponding data continuously and takes corrective action whenever needed.

3.13.3 Risk Management

The capability to mitigate risks effectively at all the different product management and development stages is critical for building a successful software product. Three types of risks can be identified, and corresponding mitigation strategies need to be devised:

- Product risks: relate to getting the product right. Product risks relate to the unique value proposition, key activities and key resources, partners, cost structure (resources) and revenue streams (pricing).
- Customer risks: relate to building a scalable path to the customers. They relate to the customer segments and channels.
- Market risks: relate to building a viable business. They relate to business measures and intersection of cost structure and revenue streams segments (determining the product's margins).

Depending on the product's life cycle stage, risks will differ. It is important to focus on addressing the right risks at the right time to minimize waste since incorrect prioritization of risks is one of the top reasons for waste.

When developing a new product, the customers' problems and existing alternatives need to be understood (problem/solution fit). From the risk perspective, aspects of the product such as value proposition and customers segments are the most important. A product manager is responsible for the risk-related activities identification, mitigation and contingency planning.

Then the product manager needs to make sure that the product being built is what customers will pay for (product/market fit). From the risk identification and mitigation perspective, aspects of the product such as unique value proposition and pricing (segment: revenue streams) are the most important. It is a huge risk if one cannot identify whether customers will pay for a product or not.

In the growth phase (see Sect. 4.4), risks associated with scaling channels and optimizing margins against competition should be considered.

Figure 3.10 can help to identify and document risks and how they can be triaged based on life cycle stage, risk level and uncertainty level. Once the risks have been triaged, mitigation strategies against the relevant risks can be described.

There is a lot of literature on risk management, e.g. [Pritch15].

3.13.4 Summary and Conclusions

Product management can become increasingly data-driven if the organization ensures the frequent availability of reliable relevant data. Performance management means the systematic continuous tracking of all business-related data, in particular KPIs, analysis and appropriate action taking. Risk Management means the systematic continuous tracking and analysis of risks related to the product, and defining and implementing appropriate mitigation strategies.

Problem/solution fit			High uncertainty
			Low uncertainty
	Low risk	High risk	
Product/market fit			High uncertainty
			Low uncertainty
	Low risk	High risk	
Growth phase			High uncertainty
			Low uncertainty
	Low risk	High risk	

Fig. 3.10 Technique for risks triage in new product development (see also [BlanDorf12])

3.14 Product Strategy Processes and Documentation

3.14.1 Overview

In this section we discuss a classification of the product strategy tasks based on frequency which can be used for a process view. We describe the structure of a product strategy document and look at tools.

3.14.2 Strategy Processes and Yearly Plan

The Merriam-Webster Dictionary defines a process as "a series of actions or operations conducing to an end". Since software product management is a continuous activity over the life cycle of a product the length of which is not predetermined, and it consists of a multitude of separate tasks, it does not make a lot of sense to look at software product management in total as one coherent process. That is also true when we only look at the combined set of activities in the product strategy column of the SPM Framework. However, it can be very helpful for product managers to classify the activities and map the periodic ones onto a yearly schedule. In that sense, we can consider the steps on the time axis as a process that aims at achieving consensus about the contents of the product strategy, even though they do not describe how the separate pieces of contents are developed or updated, and they do not cover all activities.

For classification we use the following categories:

- Continuous (C): done more often than once a month.
- Periodic (P): done monthly or less often, but with predefined frequency.
- Triggered (T): only done when a particular event or request happens, i.e. not with predefined frequency.

Strategy task	Continuous	Periodic	Triggered
Positioning and Product Definition		P for existing products	T if new
Delivery model and Service Strategy		P for existing products	T if new
Sourcing		P for annual HR planning	T if driven by current situation
Business Case and Costing		P	
Pricing		P for structural changes	T if driven by current sales situation
Ecosystem Management		P	T if driven by ecosystem members
Legal and IPR Management		P	T if driven by Sales or Development
Performance and Risk Management		P (typically monthly)	

Fig. 3.11 Classification of strategy tasks

We can apply this classification to the product strategy activities listed in the SPM Framework (Fig. 3.11):

All activities classified as P can be mapped to a yearly plan which can be used as the software product manager's workplan. The mapping is usually dependent on the schedule of activities on the business unit or company level, e.g. the yearly business planning.

3.14.3 Documentation

The product strategy describes how the product is supposed to evolve over this strategic timeframe. The table of contents can look like this:

1. Product vision.
2. Product definition.
3. Target market, potential segments.
4. Delivery model and Service Strategy.
5. Product positioning.
6. Sourcing.

7. Business plan.
8. Roadmap (see Sect. 4.1).

We recommend to describe this content in one cohesive product strategy document in order to emphasize the need for full consistency. Producing documents like this is not very popular anymore, but our experience shows that consistency suffers when a document structure does not enforce and support it. The product strategy column in the ISPMA SPM Framework contains additional elements, i.e. Pricing, Ecosystem Management, Legal and IPR Management, and Performance and Risk Management, which are also of strategic importance, but typically not fully included in a product strategy document.

All these items are highly interdependent. If, for example, business planning results in an available budget smaller than originally assumed, it will only be possible to expand the product scope to a lesser extent or more slowly. If new segments are to be added to the target market within the strategic time frame, the product scope may have to be expanded. Dependency on other products can also have considerable consequences, e.g., if certain functionalities or enabling code must be available in several products at the same time. In bigger companies that have one or several product portfolios an individual product strategy needs to be aligned with the corporate strategy and portfolio. It should be observed that interdependencies can exist on different levels of abstraction, ranging from portfolio to product to feature to function to component and also covering management and business decisions included in the strategic concerns described above.

Another way of describing elements of the product strategy is the business model canvas that we discussed in Sect. 2.4.

3.14.4 Tools

Over the last few years an increasing number of tools have become available that are explicitely intended and designed for product managers. Some provide bi-directional interfaces to popular software development tools. While they are useful for the product planning activities (see Sect. 4.5.5), there are hardly any dedicated tools for product strategy. What product managers typically use for product strategy are Office tools and increasingly business intelligence tools the more data-driven they become.

3.14.5 Summary and Conclusions

Though it does not make sense to attempt to define a process that covers all of product strategy, a process view on the consensus building through periodic activities can be helpful. We recommend documentation of the product strategy in a cohesive document. There are hardly any dedicated tools that support the product strategy tasks.

Product Planning

4

Product planning is a core activity of a software product manager as described in the Software Product Management Framework. In this book, we differentiate the following processes:

- Roadmapping: the strategic and long-range planning of how a software product shall evolve.
- Product Requirements Engineering: the collection, analysis, and documentation of the software product's requirements.
- Release Planning: the definition of the detailed contents and schedule of a forthcoming product release.

These processes are discussed in detail in a top-down sequence in Sects. 4.1–4.3. From an execution perspective, requirements management is a continuous activity while roadmaps and release plans are only revised at discrete points in time. These revisions can always be based on the knowledge that has been accumulated in requirements management and is most current. In that sense requirements management precedes release planning.

Product planning is linked to the company's planning activities, in particular portfolio management (see Sect. 5.2). Each existing product that is part of the portfolio will be in a specific life cycle stage that requires a respective management and investment focus. Product life cycle management is discussed in Sect. 4.4.

The chapter concludes with Sect. 4.5 that gives an overview of how to measure software product management performance and how to improve it.

H.-B. Kittlaus, S.A. Fricker, *Software Product Management*,
DOI 10.1007/978-3-642-55140-6_4

4.1 Roadmapping

4.1.1 Overview

Product roadmapping is a flexible technique for strategic and long-range planning of the way in which a software product will evolve [PhaFarPr04]. A product roadmap, once created, will describe what the product organization will deliver over time and how the various company functions depend on each other. Roadmapping can be applied to a single product or a portfolio, line, or family of products. Roadmaps are a structured and often graphical means for exploring and communicating the relationships between evolving and developing markets, products, and technologies over time. The roadmap translates the product strategy into an action plan how to implement the strategy over the strategic timeframe.

Roadmapping assumes that a product is in a changing but reasonably predictable environment where markets, and technologies, and products come and go. The company will evolve the product with the necessary development, marketing, research, or procurements. The roadmap describes the planned results of these activities. A roadmap is a means for documenting the predicted or planned change, agreeing with key stakeholders on the timing and scope of change projects, communicating the agreed plans, and monitoring the implementation of the plans as the activities for implementing a roadmap unfold.

A roadmap depends on corporate and product visions, on product, market, and technology research, and on product strategy. The technique assumes that the product organization has a good understanding of the company's situation and where the company should be heading. For example, product management should have a good understanding of target customers, user personas, and potential product features to satisfy the needs of these stakeholders. Marketing should have a good understanding of the current situation of the target markets, important events that the company needs to react to, and of the trends and problems that should be addressed. Research and development need to understand the capabilities and limitations of the own software and the technical contributions of third parties.

Roadmapping is best performed as an iterative process. Iteratively, a roadmap is proposed and agreed, projects launched to implement the roadmap, the achieved results reviewed, and the roadmap revised. The contents of a roadmap are proposed by exploring alternative approaches to creating value with a product and by involving the product stakeholders in decision-making. The roadmap is then used to focus requirements engineering, plan release projects, and handshake with development. During the development projects and upon a product release, the roadmap acts as a basis to check and communicate progress and to request the contributions from stakeholders that had been agreed. Deviations from the roadmap are investigated to understand their causes and the roadmap adapted to ensure relevance and feasibility. The roadmap is updated at regular intervals, before the launch of large initiatives, or upon release of software products.

Roadmapping for software products is comparable to the roadmapping of other products. Still, some specialties remain. The provision of a software product for a

customer requires hardly any elaborate fabrication process. Instead, it requires the provision of installers, e.g. for licensed products deployed on customer premises, or hosting of services, e.g. for SaaS. Roadmapping thus is unlikely to consider the design of production lines that are important for physical products. Instead, it considers deployment, integration, and provisioning of the software with an end-to-end view as suggested by DevOps [BasWebZh15]. Many software organizations also prefer implementing light-weight, agile development methods rather than elaborate, documentation-intensive methods. The way roadmapping is done should respect this preference.

A roadmap, when implemented well, allows linking the plans for developing and evolving a software product to the corporate strategy. It is a means for looking over more than just one project, thus understanding how product ideas are realized, how the development and marketing of a product come together, and how to promote relevant results and sustainability of these results. A roadmap that is agreed with product stakeholders gives the product manager the confidence that the product can be implemented. The roadmap also gives management and stakeholders transparency of how the product vision is implemented and how far the implementation has come. Such agreement and transparency are particularly important when teams depend on each other and when a change of plans implies that assumptions of what needs to be done and of what can be depended upon become invalid. The roadmap also helps to focus efforts and to identify synergies. It tells what will be done in imminent projects and what has been postponed but is likely to happen in the near future.

This chapter introduces the basic elements of roadmaps, describes the major sources of the inputs that are needed to build a roadmap, and describes approaches for building and communicating roadmaps. The reader will understand the structure and form of software product roadmaps and how to implement roadmapping.

4.1.2 Concept

In the most general sense of the word, a roadmap is a plan to guide progress toward a goal (from Merriam-Webster dictionary). The term originating from automotive travel has found increased acceptance in other areas such as science, technology, industry, and business planning to refer to a description of past events and a plan of how the future will evolve [KostScha01, Kappel01]. In software product management, a product roadmap is a document that provides features or themes of the product releases to come over the strategic timeframe. In other words, it is a plan of how to implement a product vision and strategy by building and evolving one or multiple related software products over a series of releases.

The purpose of the roadmap is to provide in bold strokes an overview of how a product will be developed over several releases, how the product is used to serve important markets, and how technology is used to build the product [Groenv97]. The roadmap is created for the product's strategic timeframe within which the product vision is to be realized by looking forward from the present.

A product roadmap usually has the following basic elements:

- Timescale
- Releases and versions
- Release themes and main features
- Target markets
- Product dependencies
- Technology impacts.

A roadmap is mostly presented graphically to illustrate dependencies between its parts. Figure 4.1 illustrates the key elements of a layered product roadmap that are used for product planning [PhaFarPr04]. The vertical axis describes the layers that are relevant for planning how to achieve the product vision. The layers used for roadmapping are selected to suit the product planning needs. The horizontal axis indicates the time. For software products with frequent releases this timeframe is up to 1 year [LeKaVä07]. Products with hardware tend to use longer timeframes [Groenv97].

A layered roadmap contains the following elements. The markets layer defines the milestones to be achieved in the product markets. Milestones refer to achievements that are important to the product's bottom line. The milestones can refer to segments that are to be targeted, differentiation to competitors, a market share that should be built, the installation base that should be achieved, contracts that are to be won, and other drivers that are used to judge product success. During the later requirements engineering, each milestone will be refined by specifying customers and stakeholders, rules and regulations, and requirements that are to be fulfilled.

The products layer contains development actions in terms of evolving product versions. Coarse-grained roadmaps indicate the timelines and themes that the

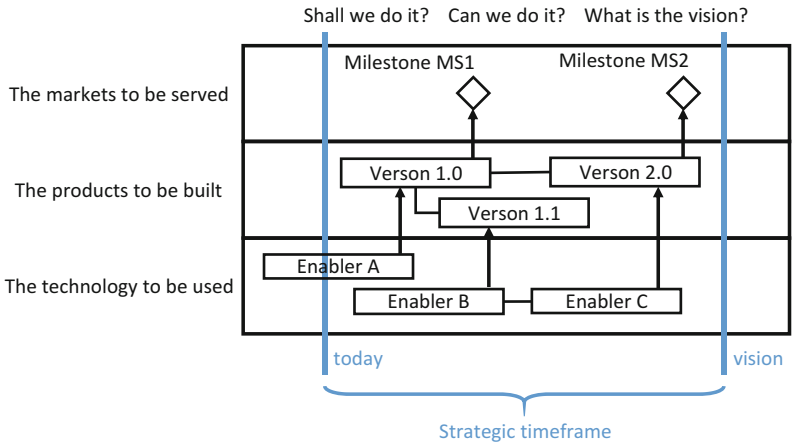

Fig. 4.1 Layered product roadmap based on [PhaFarPr04]. The time flows from *left* to *right*

software products will feature. The themes guide the scope of the product versions to be released. Fine-grained roadmaps indicate the main features to be developed. Features can further be distinguished into those that are essential for a viable product release and others that have optional character. The sequence of product development actions together with the releases intended to achieve market-level milestones reflects the strategy that the product manager pursues to succeed with the product.

The technologies layer contains important technological enablers that come and go and are used to develop or provide software products. For software products, such enablers refer to components, frameworks, standards, and ICT services. For services, milestones may be created that define quality of service or cost levels. The enablers may be developed in-house, developed in collaboration with an open-source community or partners, or procured from suppliers.

In the figure, the horizontal sequences indicate the role of the created assets for enabling future assets [KhuFrGor15]. For example, products versions may replace earlier product versions and thus are dependent on them. Special kinds of such relationships are branches from one version into multiple future versions and end-of-life milestones. The vertical cross-layer arrows indicate the value impact of created assets: the use of the assets as means to achieve the ends at which the arrows are pointing.

Some product roadmaps can be as simple as having just a product layer. Such roadmaps imply that knowledge of product impact and product technology is in the minds of those who use the roadmap. Other product roadmaps indicate additional dependencies and drivers for product development and release. To depict such dependencies and drivers, additional layers may be introduced or existing layers enhanced with additional information such as the following:

- Events of relevance for marketing such as conferences, fairs, publicized launch dates,
- Events of relevance for sales such as customer contracts or projects,
- Use scenarios and pilot projects used for product validation [FricSchu12],
- Distribution, service, and support services,
- Heartbeats with regularly spaced releases, for example monthly, quarterly, or half-yearly [RegHNBH01],
- Need for and availability of experts and capacity of product development staff [VähRau05],
- Dependencies with other product, technology, organization, or industry roadmaps, and
- Risks to be monitored and acted on.

The definition of product roadmaps requires simultaneous consideration of technology push and market pull. Technology push should be used to exploit assets that represent capabilities that are hard to copy or imitate by other companies [Teece10]. These assets should be used to facilitate and accelerate the development of products and features that provide a strategic advantage in the markets. Market

pull should be used to exploit opportunities for the company such as important tenders, cooperation with partnering customers, or synergies with marketing and sales initiatives.

A completed roadmap shows how such technology pushes and market pulls are addressed and orchestrated over time. The short-term roadmap addresses the question "shall we do it?" This roadmap is used for obtaining stakeholder commitment, launching development projects, and allocating resources to these projects. Its contents should thus be certain and indicate what the product stakeholders have agreed to do. The mid-term roadmap addresses the question "can we do it?" It is used for launching feasibility studies. Its contents should be attractive and indicate what the product stakeholders have agreed to explore with market analyses, technology evaluations, and prototyping. The long-term roadmap addresses the question "what is the vision?" It should express what the stakeholders perceive to be attractive goals in terms of market, product, and technology development. The path chosen from today into the future should enable the creation of the right market, product, and technological assets to achieve these goals.

4.1.3 Graphical Representations of Roadmaps

The format of roadmaps may vary [PhaFarPr04]. Figure 4.2 gives an overview of common variations.

The *tree-shaped roadmap* allows expression of multiple alternative time axes by depicting the growth of a tree from the root into leaves. Another name for such a tree-shaped roadmap is the feature tree. This format can be used to define alternative or complementing options of how to evolve a product. The stem represents the product's core, the branches and leaves enhancing features. A branch indicates dependencies among the nodes. The tree-shaped roadmap can later be transformed into a linear roadmap by deciding on an implementation order of the nodes while respecting the dependencies that are implied by the growth metaphor of the tree [FricSchu12].

Vector-shaped roadmaps are used to express quality or performance. The vector-shaped format allows expression of performance improvements that are achieved with a planned sequence of product releases. The horizontal axis represents time, the vertical axis performance. A vector represents a platform or technology that gives rise to a series of product versions with increasingly better performance. After

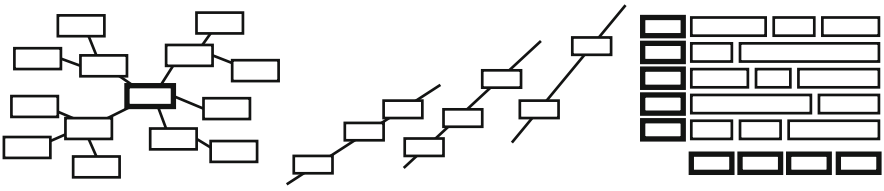

Fig. 4.2 Alternative formats of roadmaps based on [PhaFarPr04] (*left* tree, *middle* vectors,*right* bars)

some steps in the product evolution, the many changes have led to so much architectural degradation that the product cannot be evolved anymore with reasonable effort and needs replacement. At that moment, a replacement product with at least comparable functionality and performance must be available to replace the old product. Vector-shaped roadmaps are also used to analyze innovations by depicting the impact of new technologies that replace older technologies. In this context, the vector roadmaps are called S-curves.

The *bar-shaped roadmaps* represents a third important format. They are typically used for technology roadmapping [AlbKap03]. Each row corresponds to an important capability of the product and contains a description of the technologies available for implementing that capability. Such information is essential for planning improvements to the evolving product, especially when technology determines the improvements rather than the way the technology is integrated and used by the product. Such bar roadmaps may be created during technology evaluation and used as one of the important inputs to product roadmapping workshops.

4.1.4 Roadmapping Process

Product roadmapping is a creative, expert-based approach for understanding, planning, and managing product development and evolution. Product management drives the activity, supervised by senior management, and involves representatives of the product organization such as development and marketing. Important criteria for selecting the representatives are knowledge and authority in each area the participant represents. Product management must have understood the products it is responsible for, marketing its markets, and development the software and the technologies depended on.

During the roadmapping, the parties work together to build a shared understanding of how to implement the product vision and strategy and to agree on how and when they contribute to the implementation. Product management then uses the resulting roadmap to launch and supervise projects and to determine and monitor key performance indicators (KPI). Figure 4.3 gives an overview.

The first step of the roadmapping process identifies and evaluates options for how the product vision and strategy can be implemented. Each option is a building block that can be added, removed, or changed as the product evolves. Options can be alternative product concepts or complementing features that are used to enhance a base product. Options can depend on each other or be mutually exclusive. Roadmaps with a tree structure, as opposed to layered roadmaps, can be used to give a visual overview of the foreseen combinations and structure alternatives, complements, and dependencies.

Each option should be evaluated in terms of impact. The implementation of an option requires appropriate design of software architecture and processes. Each implemented option creates value by satisfying needs and expectations of stakeholders, by generating customer and user desires, and by establishing assets

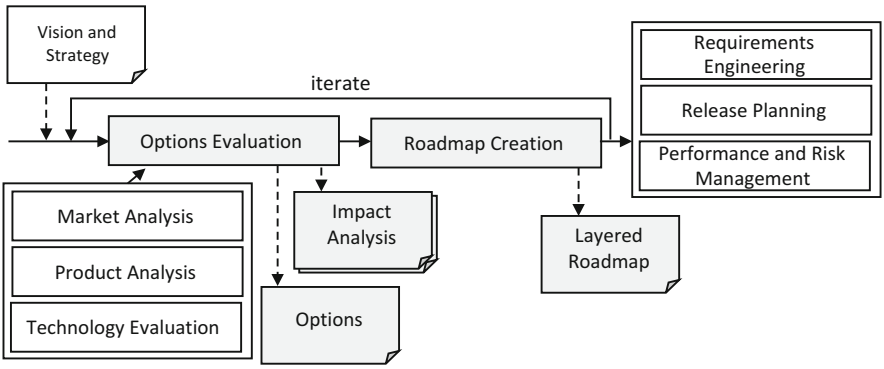

Fig. 4.3 Roadmapping process. *Grey*: core activities, *white*: dependencies

Attribute	Description
Title	Name of the option (product concept, feature, or theme).
Description	Specification of the concept, feature, or theme.
Value	Supported market needs, stakeholders, and elements of vision and strategy.
Concept	Specification of planned implementation, including components, frameworks, services, and knowledge needed for implementation.
Evaluation	Strengths, limitations, and risks of the choices documented above.
Alternatives	Alternative concepts that have been dismissed and why they were dismissed.
Estimates	Value and cost estimates.

Fig. 4.4 Documentation of an option [ClaHeyScho08, FrGoBySc10]

that can be used to build enhancements and future products. Each option requires time, staff, capacity, and financing to implement and deliver it to customers. Options may also affect the creation of social capital with knowledge, skills, and relationships that are to be built [KhuGorW13].

A pragmatic way to evaluate options is to discuss them with the stakeholders. Such a dialogue can happen during planning meetings [Pichler10] or in a more elaborate handshaking process that leads to implementation proposals [FrGoBySc10]. Figure 4.4 shows a lightweight template of how the results of these discussions have been documented and agreed with marketing (value attribute) and development (design attribute). Some organizations implement such handshaking as a continuous process.

The second step of the roadmapping process is the creation and agreement of the roadmap. A layered roadmap helps to define when and how each option will be realized or whether the option is discarded in favor of others. This step should be launched as soon as the options and their impacts are sufficiently understood. The step was effectively implemented with a workshop approach where decision makers of product management, marketing, development, and other product stakeholders participate [PhaFarPr07].

During the workshop, the roadmap is built and reviewed by roughly following the following agenda:

- Introduction: introduce workshop objectives, participants, and agenda.
- Strategic landscape: discuss the product vision and strategy, the situation, opportunities, and issues of the company, markets, and technology. The aim is to build a shared understanding of product context. Important events, issues, and activities are marked at the appropriate place.
- Options: discuss and prioritize the known product concepts and features to achieve a shared understanding of options, dependencies, and impacts.
- The way forward: use the roadmap to explore possible sequences of product development and to review dependencies with Marketing and Development. Conclude with an agreement of what shall be done with imminent projects and what shall be explored with feasibility studies.
- Consensus: agree on actions and assess satisfaction with the workshop results.

The roadmapping workshop delivers a plan of how a product will be developed and evolved and a forecast of what will happen in the context of the company and how the product team will react to these. The selection of the workshop participants ensures that the company uses best-possible knowledge for doing these predictions. The contributions of the key product stakeholders in the joint decision-making ensure understanding of the roadmap and commitment to implementing the actions necessary to realize the roadmap. For product development, the roadmap is a means to facilitate technology push. For marketing, it is a means to achieve market pull.

The results of the roadmapping workshop depend on the knowledge of the participants. Much that is discussed and agreed may not be feasible or may not generate the desired impacts. Technologies may not be as effective as anticipated and have undesirable side effects. Customers may not accept a product, for example, because base factors such as security, privacy, quality of service, or the user experience turn out not to be as good as needed. It is thus essential for the product manager to evaluate product use, product design, and market acceptance early. Lessons-learned from ongoing projects, feasibility studies, product monitoring, and customer feedback are key inputs for the next roadmapping iteration.

4.1.5 Variations of Roadmapping

Product managers may use roadmapping as a tool not only for product planning but also for other purposes. Product roadmaps can be used as a tool to achieve synergies in a portfolio of products for cost reduction and for convincing and coordinating parties external to the company. The former type is called cross-product roadmaps, the latter external roadmaps. For both alternative purposes, the product roadmap sketched in Fig. 4.1 is used as a starting point and adapted.

Cross-product roadmaps are used when several products are released with the same technological base, within a product line [PoBoeLi05]. The product line, as

defined in Chap. 2, is a form of reuse that focuses on identifying the common and variable parts between multiple related products. The common part is often called a platform that is shared by the products. To plan reuse of the platform, a feature tree is created for the whole set of products that are contained in the product line [SchoHeTB07]. The feature tree describes the options available to build the products for specific markets or customer segments. In a second step, a layered roadmap is created that defines the when the platform and the derived products are to be developed. A cross-product roadmap differs from the basic already introduced product roadmap in that it contains multiple products and is used to analyze the opportunities for planned reuse among the products. Once created, the roadmap is used to coordinate the actions provided by one team to another.

Cross-product roadmaps may also be used by organizations that offer a portfolio or family of products, but are not interested in systematic re-use. In this situation, the cross-product roadmap is used to define how the products complement each other. It is also used to ensure that the products contained in the family are non-overlapping and that the family does not have any gaps in the product offering.

Finally, cross-product roadmaps may also be used to plan and manage the evolution of a software ecosystem. Such use of roadmapping has similarities with industry-level roadmap [KostScha01], where multiple organizations convene and agree on how to cooperate to address important industry-wide problems. Such use of roadmapping requires a joint objective and willingness of the companies participating in the roadmapping effort to share knowledge and to cooperate.

External roadmaps, in contrast to the roadmaps used internally in a company, are used for allowing customers, market research agencies, suppliers, and other stakeholders external to the organization to understand the product vision and strategy and the approach initiated to implement them. Such sharing of a roadmap allows positioning a product and eliciting feedback that can be used to validate the plans for product development and evolution. External roadmaps also play an important role in demonstrating the viability of a product. They are used to build trust in the commitment of the company to long-term continuous investment in the product. Influential customers or partners may want to see a product roadmap, against the signature of a non-disclosure agreement, before they make a significant investment decision or decide about continued cooperation. Similarly, press and market analysts base their judgment of where the company is heading on a convincing story about a product's future expressed in the roadmap.

External roadmaps are derived from internal roadmaps and contain just enough information to fulfill the information need without disclosing too much of the company's confidential internals that they wish to keep secret. To influence external stakeholders, the external roadmap should include information that one would use for competitive analysis when defining one's product strategy. Information should be included about the targeted vision and strategy, current and announced products with differentiating features, supported applications and solutions, and sometimes early pricing information. So should information about strengths of the firm, including an overview of the customer base, market share, distribution channels, capabilities that are hard to copy or replicate, and important partnerships. The goal

of such information is to allow customers and partners to build trust in the viability of product and company and to encourage others to refrain from competing. However, revealing too much can lead customers to postpone the investment and wait for the next release or version. And it can over-promise, leading to disappointment later if development plans change.

Attached to the external roadmaps are legal disclaimers that explain the confidential and legally non-binding character of the information. The former contributes to the credibility of the roadmap. The latter allows product management to retain flexibility while protecting the company against litigations. Such flexibility is important when the roadmap has become invalid, and changes are necessary. Companies have to decide how they want to deal with the confidentiality of a roadmap. Some companies require external parties to sign non-disclosure agreements before the roadmap is presented to them. Other companies make their external roadmap freely available on their respective web site.

External roadmaps can also be used to coordinate product development with customers, partners, and other external stakeholders. When an external roadmap is used for that purpose, the external parties should be involved in its creation. The external roadmap is then used to constrain the internal roadmap. Such roadmap creation follows a process like the one outlined in Fig. 4.3. It leads to the same benefits of tapping into the knowledge of all involved participants and committing the partners to joint actions. The external roadmap then has the character of an industry roadmap that is centered on the development of the product's ecosystem. The cooperating partners are responsible for keeping their internal roadmaps consistent with the jointly agreed roadmap.

The term roadmap is also used with agile development methodologies, but with a different meaning. In SAFe [Leffing16], it stands for a near-term plan of deliverables, e.g. what Development will produce over the next 6 months.

4.1.6 Summary and Conclusions

This section has given an overview of the roadmapping concept and how it is used for product planning. It has described the layered roadmap for implementing the product vision and strategy and introduced additional forms of roadmaps that are useful for defining planning options and reflecting upon them. The section has also described a workshop-based roadmapping process for developing an internal roadmap and explained the relationship of internal roadmapping to external roadmapping.

Once created, the roadmap is a tool for communicating how the product vision and strategy will be implemented. In comparison to an early vision statement, the roadmap is more detailed, describes the impact on development and marketing, states concrete actions that will be performed by the important product stakeholders. The short-term roadmap provides justification for committing resources to the projects that need to be launched to develop and evolve the product. The mid-term roadmap captures the technology push and market pull ideas of the

product stakeholders. It defines the research questions that need to be investigated with feasibility studies. Finally, the roadmap specifies the sequence and scope of projects in coarse-grained terms. This information helps the product manager to launch the right projects, orchestrate the parties that need to cooperate, and monitor the progress of the developing and evolving product. The elements of the roadmap refer to the goals that need to be achieved, the milestones for goal achievement, and the dependencies affected when milestones or goals are not met.

The roadmapping results provide inputs for dependent product management activities. The identified activities provide the basis for forecasting, budgeting and the instantiation of projects for the development of specific product releases. The scope of the planned releases provides information about what kind of requirements engineering needs to be performed, including user interfaces that require user interaction design, interfaces towards external software systems, and other features that require refined alignment between markets and technology. The roadmap is also a starting point for the detailed planning of upcoming software releases. Release planning will complement the roadmaps with detailed prioritization and definition of minimal versions and enhancements of the selected features. The roadmap will also provide essential information for defining or refining key performance indicators such as the timing and quality of the product versions and the impact of these versions on the market. As for any other decision-making activity, the relationship between roadmapping and release planning is neither top-down nor bottom-up. Instead, one influences the other. Thus, the roadmap should be reviewed when important insights are gained from the dependent activities.

4.2 Product Requirements Engineering

4.2.1 Overview

Requirements engineering is the process of eliciting needs and expectations from stakeholders, identifying concepts to satisfy these needs, and validating whether these concepts satisfy the needs. The resulting requirements are specified with appropriate notations, communicated to the staff or vendors who implement the requirements, and used to check whether the delivered product conforms to the requirements. The alignment between product concepts and stakeholder needs is essential for a product's value creations and to win the support of the stakeholders.

This section introduces important requirements engineering concepts, describes requirements engineering methodology for the four important product scenarios (see Sect. 2.5), outlines how to document requirements and how to manage the requirements engineering process.

4.2.2 Concepts

According to the IEEE standard glossary of software engineering terminology, a requirement is "*(1) a condition or capability needed by a user to solve a problem or achieve an objective and (2) a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents*." Applied to product management and stated in more straight-forward terms, requirements are characteristics of the software product that are (a) needed by stakeholders of the software product and (b) agreed with the product team or supplier. Requirements may also be imposed, for example, because of market regulations. The requirements definition will guide the activities that are to be performed in the requirements engineering work.

For a software product, a *stakeholder* is a person, group of people, or organization who influences the product. Product stakeholders include parties outside and inside the product organization. Outside are customers and users as well as channels used to deliver and service the product. User profiles are often characterized by so-called Personas [PruGru03]. Inside are corporate management and the company functions that depend on the product, including marketing, sales, distribution, service, and support. Stakeholders are also representatives of dependent systems and other products that the product interfaces with. Regulatory bodies that the product must comply with are also considered to be stakeholders. Even though the number of stakeholders is high, the product manager does not need to involve all of them in requirements engineering. The involvement of stakeholders that have the knowledge to make the product attractive and stakeholders who have the power to implement or stop the project are both critical to the process.

Software requirements are *functional* if they define what the software does in reaction to inputs. According to ISO/IEC FDIS 25010, *quality* requirements are the "-ilities" that describe the performance, compatibility, usability, reliability, security, maintainability, and portability of the product [Glinz08]. Requirements about product packaging, licensing, delivery, and service requirements complement the product requirements. For many of the requirements, tests will be developed to verify that the developed software product indeed carries the characteristics.

A possible product characteristic is only a requirement if at least one influential-enough stakeholder needs it. Otherwise, it is merely an idea. This differentiation between requirements and ideas has important implications on the requirements process: the results from creativity workshops and related activities should only be considered as product requirements when enough stakeholder support has been gained. If the need for an idea is not confirmed by stakeholders, the idea should be treated with caution.

Let's use the Samsung S Health product[1] to exemplify the concepts. A functional requirement of such a lifestyle application is to provide the user with the ability to count steps. In response to movements of the smart phone, the app increases the step count that it measures. A quality requirement is the precision of the step-counting. That precision could be measured, for a given movement sequence, in terms of correct, wrong, and missed step counts and the quality requirement expressed in those terms.

A common quality requirement is interoperability with other applications. In the case of Samsung S Health, interoperability is implemented via the Samsung Digital Health SDK API with functions that offer access to health data and S Health services. Other product requirements refer to the contents and design of the product home page that is used for product documentation.

The integration with an electronic patient health record is an idea that many vendors of health and lifestyle applications are discussing. This idea should only be considered as a requirement for implementation, however, if support from medical personnel and patients can be obtained that require the integration.

Product managers not only manage requirements but also inputs on other abstraction levels, such as goals and constraints for product design [GorsWohl06]. The IEEE definition is useful for differentiating between these concepts. Requirements are characteristics of the product. A *goal* describes the desired impacts of the product. Goals are as important as requirements because goals describe the value proposition of the product. According to ISO/IEC FDIS 25010, common goals include desired usefulness, trust, pleasure, and comfort for users, effectiveness, efficiency improvements, freedom from risk and flexibility for customers, and compliance with regulations. Requirements engineering is concerned with eliciting the goals that the product should achieve and the requirements the product should implement to satisfy the product goals. The explanation of how requirements will allow achieving the goals is called *vertical traceability*.

Product managers are not only confronted with requirements and goals, but also with *constraints* for product design. Design differs from requirements in that the design decisions are characteristics of the software product that are *not* visible for a stakeholder. Usually, design does not need to be agreed with stakeholders but is under the responsibility of the product organization. Still, for some design decisions, negotiations cannot be avoided because stakeholders impose them as *constraints* on the software product.

[1] https://shealth.samsung.com/

For Samsung S Health, one of the goals is a "healthy lifestyle" for the user. The goal is not a requirement because it describes the impact of product use on the user. The adoption of a healthy lifestyle is clearly a characteristic of the user and not of the product. The S Health product will contribute to achieving that goal by encouraging the user to adopt such a lifestyle.

In the example of S Health, a constraint may be raised by the Cloud operations group of Samsung that requires the use of Docker for flexible deployment of the S Health backend. That constraint is justified by efficiency and flexibility goals of the Cloud operations group and the ICT technology stack on which the S Health backend is deployed. Again, the arguments of how design decisions are connected to requirements and goals are called vertical traceability.

Product managers are not only confronted with product requirements but also with customer and project requirements [Schien02]. The three types of requirements can be understood by looking at the flow of information about needs and expectations from the customer via the product manager to development.

Customer requirements express needs and expectations of a customer towards a software solution. In customer-controlled installations, customer requirements are engineered in customer-specific projects that deliver a software system integrated, deployed, and configured specifically for that customer. Such a project uses the customer requirements to tailor the vendor's software product and integrate it with other systems of the customer. In vendor-controlled installations, the customer uses the provided software service largely as-is and use his customer requirements for configuration.

Product requirements represent a consolidation of the requirements of individual customers. With product requirements, the product manager considers the needs of a market from the perspective of product the strategy. When consolidating customer requirements into product requirements, the product manager looks for generalization across customers, "*which requirements are common enough to be considered for the product?*", and alignment with strategic objectives, "*which requirements can be justified with the product vision and strategy?*" The link between the product requirements and the customer requirements that the product requirements originate from is called *horizontal traceability*. The product requirements represent the pool of requirements that a product manager considers for release planning. The more the markets have been served with the product, the more product requirements are available to the product manager.

Project requirements are a selection of product requirements for implementation in a product release. The development project staff takes responsibility for the implementation of the selected requirements and translates them into a specification of a new software release. In contrast to the product requirements, that specification is detailed and precise enough for implementation and testing. The link between the

project requirements and the product requirements that the project requirements originate from is called horizontal traceability again. In an agile environment, where a project organization is not utilized, we suggest to use the terms "development requirements" or "team level requirements". These are the requirements in the teams' backlogs for which the respective product owner (in Scrum terminology) is responsible.

A hospital that is interested in using S Health for lifestyle measurement of patients may request the integration of S Health measurements with the patient records stored in the hospital information system. This request is an example of a customer requirement. A recurring need of integrating S Health with various hospital information systems would encourage the S Health product manager to look for standards that can be used to specify and offer a generic application programming interface (API) that eases such integration.[2] A decision to build such an API is an example of a derived product requirement. Once a decision is taken to implement the product requirement, the API becomes a project requirement. The development project for S Health v4.4.0.0119, for example, included requirements for a new cycling speedometer and requirements for connectivity with the Garmin Bike Sensor.

4.2.3 Requirements Engineering Methodology

Requirements engineering is the systematic process of aligning a software product with stakeholder needs. Without such alignment, the software would be a mere collection of bits and bytes that would not create any value, thus would not have any chance to be sold. We consider the alignment as a process of seeking inputs from the market to understand needs, adjusting the product to meet the needs, and checking whether the changed product leads to improved satisfaction. The better the alignment is the better the performance of the product will be.

One of the product manager's core tasks is the management of product requirements. The product manager is responsible for the following tasks continuously and before a release project:

- Consultation with experts and employees, creativity to generate ideas for the product, validation of these ideas by exposing them to stakeholders in discussions or via prototypes.
- Collection of requirements from diverse sources. Customer projects and ideas from development constitute important sources. Other inputs can come from

[2]Samsung offers such an API on http://developer.samsung.com/health

user representatives, market research, and developers, legacy and competitors' systems, laws and regulations, and previously used requirements specifications.
- Functional analysis, including clarification and consolidation of the requirements and estimation of value. Such analysis is particularly important when products do not stand alone but form part of a larger offering.
- Technical analysis to describe how the requirements may be implemented and what the estimated effort and cost are.
- Documentation of the results of the functional analysis with an appropriate level of detail as a base for development.
- Go/no go and make/buy decision according to the company-specific decision processes.

During the development of essentially every product, there will be the need to change existing requirements or include completely new ones. Such changes may pose a risk for a project and possess a potential for conflict between Sales and Marketing, Development, and Product Management, which the product manager will need to resolve. It is important in this context to understand how requirements management and the development process are interconnected, since different development methods also differ in how much they embrace requirement changes. The product manager is responsible for the following tasks in the implementation of a product release:

- Implementation management by tracing the individual requirements, i.e. by documenting where and how each requirement is implemented and can be used for customer feedback.
- Change management by deciding on change requests that lead to new requirements and changed or dropped existing requirements and documentation of the rejection or acceptance of each change request with the rationale for the decision.
- Quality assurance, if possible through an independent QA organization, to verify that the implementation is consistent and complete with respect to the original requirement.

The product manager has the choice between two tactical approaches for requirements engineering. The traditional approach involves *inquiry cycles* [PohlRupp11] with the following elements:

- Elicitation is the systematic application of methods such as interviews, focus groups, workshops, observation, creativity, surveys, and artifact analysis to understand the application domain and to identify stakeholders with their objectives and expectations. Important sources of requirements are internal; in particular research and development can ensure innovation. The resulting requirements need to be properly documented, either in natural language or model-based.

- Triage means a first analysis and decision if a given requirement is inside or outside of the product scope, i.e. needs more detailed analysis or not. This may include a bundling of small low-level requirements in order to increase the productivity of the process. The product manager will typically maintain a backlog with a significant number of requirements that are inside the product scope, but not yet implemented, and that need frequent reevaluation.
- Analysis of the remaining requirements aims at matching the elicited information with appropriate solution concepts by applying specification, modeling, and prototyping methods which requires decision making. It includes estimation of cost and required resources.
- Selection means the decision making which requirements are implemented in a particular product release, i.e. selection includes prioritization and is part of release planning.
- Validation involves the application of methods such as reviews, inspections, and simulations to ensure that the proposed solution adequately takes the problem context into account and is acceptable to its stakeholders. The result of successful requirements engineering is a specification that documents agreement between stakeholders and development or suppliers of what will be delivered.

Such inquiry is typically performed in situations where the customers are in control of the execution environment of the software product. More recently, *experimentation* has become an attractive alternative. It may be understood as the development of hypotheses about possibly attractive products and features and experimentation to test his hypotheses. This alternative approach has become attractive with the maturation analytics and feedback tools that support such research.

Several requirement life cycle phase models exist. Examples of phases that might be used are:

- *New*: the initial state of a requirement.
- *Approved*: the requirement is approved and ready to be analyzed.
- *Specified*: the requirement has been analyzed with regard to implementation concept and cost and impact estimates.
- *Rejected*: the requirement is a duplicate, already implemented, or out of scope.
- *Selected*: the requirement has been selected for implementation with a given priority.
- *Implemented*: the requirement has been implemented, i.e. development is finished.
- *Tested*: the necessary tests have been carried out in order to ensure an adequate level of quality.
- *Released*: all activities for the product release have been completed.

However, the naming and number of phases may vary, depending on the preferences of the organization and the product scenario.

Requirements engineering may be performed at the level of the whole product, at the level of features, or at the level of a software release. The product level is essential in the powerboat situation, where the product manager uses experimentation to align a minimum viable product with market needs. The feature level is essential in the speedboat situation where the product manager uses experiments for identifying and defining complements to the minimal product that is frequently released. These complements are intended to make the product attractive to new market segments. The release focus is essential in the icebreaker and cruise ship situations, where the product manager decides about a well-designed scope of a product that is released comparatively rarely.

The following subsections introduce common requirements engineering methods. These methods are presented in the product management scenario (see Sect. 2.5), into which they fit best. However, each method may be used in any of the situations if it leads to desired knowledge about product-market alignment.

Powerboat

The powerboat situation is characterized by a lack of knowledge about customer needs and how a product could satisfy these needs. Here, the product manager has the role of the creative researcher that rapidly needs to identify an attractive product-market fit. To minimize complexity and risks of product development and to achieve fast time-to-market, the software product is on purpose kept as small as possible. A minimalistic but still useful product is called *minimum viable product*. Once a product-market alignment is found, the product is offered to the market and the marketing and sales processes are started.

In the powerboat situation, the goal of requirements engineering is to innovate. The organization wants to learn about the market and its needs and to identify a product concept that is perceived as attractive. The requirements engineering process starts with an idea of what an attractive software product might be, prototyping the product idea, and validating the prototype with recruited market representative [Ries11, Maurya12]. The process succeeds if the idea is simple to implement and attractive enough to make first customers curious and entices them to participate in validation and elicitation activities. Their feedback is used to improve the product and to make it even more attractive to a larger potential market segment. The process repeats until an alignment has been found with a market segment that can generate a business volume large enough for launching product operations and growth.

Startups have been using rapid iterations of product ideation, prototyping, and validation in a process that others call pivoting [Ries11]. Pivoting is not exclusive for start-ups and can be easily implemented also by product managers that want to explore ideas for products whose operations are under their control. Critical for good results of the pivoting process is the flexibility for abandoning early product ideas and replacing them with ideas that are clearly superior. The selection of attractive target markets and appropriate sampling of these markets is critical so that the elicited customer and user feedbacks are relevant and represent the largest

potential for product revenue and growth. Common sampling strategies used in such qualitative research are [MileHub94]:

- Typical case or random sampling, where representative customers and users are invited (the best results are achieved by selecting the invitees randomly),
- Critical case sampling, where lessons from one representative can be extrapolated to a whole market segment,
- Maximum variation sampling, where the boundaries of a market segment are explored, and
- Negative cases, where the product idea is not expected to be successful.

Convenience samples, where colleagues and friends are invited, or snowball samples, where participants suggest other participants are both to be avoided. These samples would lead to results that are biased and do not represent the market well. That is also why samples need to be changed over time. Sampling can conclude when saturation has been observed, i.e. when the results obtained in workshops already have been known in advance and do not add substantial new knowledge about customers and their interaction with the intended product.

The powerboat situation calls for creativity, knowledge, and ways to make the product tangible to obtain feedback. Requirements engineering techniques that are suitable for this situation are prototyping and workshops. These techniques are outlined in the next few paragraphs.

Product ideas are created by encouraging people competent in the product domain to think out of the box. A really new idea solves problems that have not been solved so far or solves a well-known problem with a new concept much better than previous products did. Good ideas are often relatively cheap to implement for the product company and generate a “wow” for customers and other stakeholders. Good ideas may be created by combining for the first time a well-known problem with a well-known solution. Such re-combinations may be identified in workshops where the selected participants brainstorm [MaiGiRob04]. Many creativity techniques have been developed to support such workshops, including the 6-3-5 technique, where six participants handing off three ideas each to each other five times, the analogy technique where solutions from one domain are transferred to another domain, and the Six Thinking Hats technique that allows evaluating ideas from the perspectives of facts, value, risk, emotions, possibilities, and structured reasoning [PohlRupp11].

Large companies may have research departments to produce innovations. Alternatively, a software vendor may work with universities. Researchers can offer a product manager important ideas and direction, especially for product strategy and further technological development. Even if a lot of research ideas cannot be directly converted into products, research can provide insight into new developments which can be expected in the next 3–5 years, and to think about how current products can be positioned accordingly and early enough. Also, many innovations can be implemented faster and with better quality if they are initially prototyped and tested in a research environment. To enable such a pre-product-development phase,

researchers will have knowledge of relevant product technologies and of experimentation methodologies that product development departments often do not have.

Prototypes can be created at varying levels of fidelity. Important for a good prototype is that the idea becomes clearly visible and that the relevant stakeholders can give feedback on the implemented idea once they have experienced it.

- Early-phase low-fidelity prototypes: user interfaces can be approximated by drawing the wireframe of the graphical user interface [MivBen14]. Such drawings can then be combined into a sequence of screens that show the appearance of the application for a scenario of using the application. Products that involve hardware can use existing devices that carry intended functionality but may not look exactly as intended or vice-versa. The key is rapid development and not fidelity. Such early-phase prototypes make a software tangible; their low fidelity encourages feedback and discussion.
- Mid-fidelity prototypes: for maturing the early prototypes, tools may be used to render interfaces that the user can interact with by clicking on the GUI components (click-dummy prototype). Hardware may approximate a final look-and-feel with the help of 3D printers that have become capable of creating a large variety of shapes and functions.
- Late-phase high-fidelity prototypes: as a next step, the target development technology can be used to implement the look and feel with a high level of fidelity and with an early version of the final functionality. These late-stage prototypes are useful for letting users explore the use of an application. The high fidelity enables validation of the ideas it implements in realistic conditions that have been experienced using the prototype.

Product ideas can be validated in workshops that involve a review of the vision by the stakeholders affected by the product (especially customers and users), role-playing the product, and debating the benefits, limitations, and risks of the product. Such focus groups produce valuable inputs for the design of the software product because they elicit knowledge and experiences about the use context of the product from the perspective of the future customers and put these results into the perspective of other customers that may modify their importance or interpretation [KruCas14].

For role-plays to be effective, it is critical that the participants be true customers and users, that the role-play take place at the location and setting where the product will be used, and that the moderator help the participants to walk through the intended usage process. The role-play will generate emotions about the product, both positive and negative, and discussions of how to exploit the product and how to solve problems that have not been thought about during ideation and prototyping. This feedback can be turned into a backlog of how to adjust the product to make it more attractive or requirements for selection of user and customer groups to whom the product should be offered. If video-recorded, the workshops also generate ample footage for requirements communication to developers who cannot participate [FricSFT16] and for early marketing and publicity with product videos.

Kano Model: Attractive Attributes (Delighters)
Powerboat product ideas are commonly intended to be innovative. Customers often did not anticipate such solutions, thus were not seeking them in advance. It is the innovating product itself that generates the desire for the innovation and the impetus to take advantage of the new product. The Japanese researcher Kano suggested that such products or features are called "delighters" or "attractive attributes" [KaSeTT84]. The implementation of a delighter leads to a "wow" on the customer side because it can result in great satisfaction. The lack of a delighter will not be noticed.
A delighter X can be identified in a product survey with very positive answers to a question like "How do you feel if you would obtain a product with capabilities X?" and neutral answers to a question like "How do you feel if our product would not have capability X?" The product manager benefits from such customer surveys by obtaining information about a market. For example, if customers felt indifferent to the first and second questions the product idea would be judged unattractive and should be discarded.

Speedboat

The speedboat situation is characterized by the presence of a first product that is aligned with market needs and that the product organization wants to grow. How to achieve that growth is unclear, however. The product manager has been working with a few enthusiastic or visionary customers and may not understand the needs and expectations of the large majority of pragmatic more conservative customers yet that represent the mainstream market [Moore14]. This re-alignment that some call "growth hacking" can be addressed by scaling up the Powerboat requirements engineering activities: to reach new customers, the product manager changes the positioning of the now growing product and thereby accepts significantly new requirements.

The speedboat situation calls for ideas for new product features and feedback from the market about how attractive they are. Requirements engineering techniques that are suitable for this situation are idea castings, market research, user groups, input from development, Hackathons, and "growth hacking." These techniques are outlined in the next few paragraphs.

Ideas for how to evolve the product are one of the important inputs in the Speedboat situation. New features are needed that make the product attractive to customers that have not shown interest in the minimally viable product until to date. At the same time, many people have developed product expertise and should provide inputs and implementation of product changes depends on even more. An approach to harvesting ideas from these people are idea castings [GoFrPaKu10]. A casting follows the steps of spreading calls for ideas in the organization, inviting for meetings where employees to share their ideas with product management, and

combining the ideas into a product concept with a strong business case. Such idea castings are especially effective when the successful idea owners are rewarded by becoming part of the product organization, for example with a leading position in product development.

To obtain inputs from the market side, market research and user groups play an important role.

- Market research, introduced earlier in this book, offers aggregated market analysis with feedback on product strengths and weaknesses and trends of how the markets and technology develop.
- User groups, in comparison, offer a primary source of individual inputs. A user group focuses on exchanging information between its members and the software vendor. That exchange is intended to allow the members to influence the vendor's product development. For the product manager, this is an excellent tool for obtaining first-hand feedback about the strengths and weaknesses of his product from the perspective of the users and customers. Most major software vendors have user groups or, the larger variant, conferences at the international scale. For example, SAP runs its SAPphire conferences each year in collaboration with local SAP User Groups.

A phenomenon that is difficult to manage in this context is the tendency of executives to match or just outdo the competitor of the moment. Such "shooting behind the duck" will fail utterly to consider that the competitor will move while development takes place. A product manager must anticipate what customers want beyond that current competitor offerings and get there before the competition.

To obtain technology-oriented inputs, product management collaborates with the development organization. No other unit has a deeper understanding of the technical details. There are numerous requirements that derive from technical necessity and must be implemented. An example is changing to a new operating system or database release—something that may not add value to the product but may be necessary, e.g. because maintenance for the old version is terminated or because the new version offers better performance or flexibility. When considering inputs from development, however, the product manager needs to be aware that sometimes a simple technical solution that is available in a timely fashion is often better than a technically interesting, and in some sense perfect or "gold-plated" product.

An increasingly approach to explore technology are hackathons, events in which developers meet peers, technology suppliers, and other stakeholders [RaaMoBia13]. For the organizer, the goal of the event is an efficient assessment of the requirements and design of products and platforms that guide future development. Participants benefit from the belonging to a community that results from the collaboration, inspiration for product development, and motivation to drive the personal work forward. Hackathons are particularly well suited for exploring products, platforms, and standards that target developers and depend on interoperability.

Looking outside the company, social media have become an important source for identifying topics and influential individuals or organizations that should be involved in requirements engineering. Twitter, for example, has several hundred million users that are active at least once per month. To support social media analysis, applications have been developed to detect emerging topics and trends [MathKou10]. Social media further provides possibilities to identify interesting individuals and organizations that can connect important topics that are unconnected otherwise. These so-called structural holes have been shown to generate great product opportunities [LinWuWen12].

A related set of techniques has started to establish itself under the term "growth hacking." Startups use this term to refer to the combination of creativity, analytical thinking, and social metrics to improve product exposure and sales. Techniques that are being used aim at improving product sales and obtaining feedback at a low-cost, including search engine optimization and content marketing. A variety of techniques can be used for monitoring product use and obtaining feedback from customers to guide product evolution:

- Analytics [CrolYosk13]: the collection of measurements about the use of a website or application screens. Particularly interesting are frequencies of page or screen use, click-trails, and measurements of performance, reliability, and related product qualities.
- Experiments [McFarland12]: selective presentation of content variations to learn about user preferences. A widely-used approach is A/B testing, where two alternatives are compared with each other.
- Micro surveys: the presentation of forms that allow users to provide feedback. Commonly, questions are asked about the user's quality of experience, bugs that have been encountered, and ideas for improving the application. An interactive chat between the user and the vendor's sales staff or help desk may enhance a survey and the knowledge obtained from it.

No matter the source of inputs, the product manager must view the received inputs in an overall context. He must know how representative the suggested ideas and needs are. The inputs that are presented most vehemently are not always the most important ones or the ones that will promote or secure the product in the market in the medium and long term. Given the finite resources to develop and improve the product, the product manager will need to be careful in choosing the requirements.

A software product has not only to deliver functionality, but needs to satisfy non-functional requirements in addition, also known as quality requirements. Quality attributes that are of particular concern for vendor-controlled products are:

- Usability: Usability is a key success factor for essentially all products. Usability includes being easy to learn, consistent design of the user interface (or APIs), understandable error messages, searchable, indexed help functions and documentation. In a vendor-controlled situation, usability is especially critical as the

users have low switching cost and will seek the product that is most attractive to use.

- (Un-)installability: many vendor-controlled products have mobile or PC front ends that must be installed and replaced, e.g. as part of a product update. Similarly, web-based applications may require the installation of runtime environments or browser plugins. Especially for consumer software, the user expects a smooth installation within a very short time, i.e. minutes. Not meeting this expectation means frustration and dissatisfaction.
- Documentation: Even though customers may not read product documentation and expect to be able to install and use a product intuitively, product documentation continues to be a significant quality aspect. Documentation may be presented in the format of a manual or help indexes or take the format of videos that introduce the installation and use of the software or a feature.
- Maintainability: Maintainability is primarily an internal objective for a software company. It can be achieved by thorough and current documentation of the results of development, i.e. architecture, high and low level design, and code plus the conformance to development standards like naming conventions, coding standards, in-code-comments, and interface descriptions. Maintainability will be influenced by decisions about the use of development frameworks, platforms, and components that, for example, may have been developed and matured by open source communities.

Kano Model: One-Dimensional Attributes (Satisfiers)

The product features that are elicited, tested, and implemented in the speedboat situation represent characteristics of software products for which better fulfillment leads to an increment of customer satisfaction. Each customer segment expects such characteristics to be present in the product; the absence of these characteristics in the minimal product variant developed in the powerboat stage was one reason to forgo the procurement of the product. Kano suggested that such product characteristics are called "satisfiers" or "one-dimensional attributes" [KaSeTT84]. The more comprehensively the satisfiers are implemented, the more attractive the product is for the target customers. The fewer the satisfiers, the less the product is attractive for the customers.

A satisfier X can be identified in product surveys with positive answers to a question like "How would you feel if you obtained a product with capabilities X?" and negative answers to a question like "How would you feel if our product did not have capability X?" Satisfiers give the product manager information for prioritizing development. Strong negative reactions to the absence of features paired with strong positive reactions to the presence of

(continued)

these same features indicates where to set priorities for product enhancements. Similarities among respondents point to customer segments that can be addressed with a product release containing those features.

Icebreaker

The icebreaker situation is characterized by the presence of customer environments into which a new software product will be deployed and integrated. The customers are responsible for the business processes that are executed by the software users and for the operation (hosting) of the product. Consequently, each deployment of the product is a custom project, where customer-specific requirements are elicited and used to adapt the software. Product requirements engineering consolidates these customer requirements and decides whether they are included in the product or are relegated to individual customer projects.

The integration of a system in a customer's environment can be motivated by business process changes that allow the customer to improve productivity or create more revenue [Harmon14]. Such changes may be of process improvement nature and concern specific changes to a process that, for example, may be enabled by the vendor's product. Six Sigma is a well-known example of a method framework for improving business processes. More drastic changes to a customer organization may involve the design or redesign of whole business processes. The introduction of automation in a process that was implemented manually could be considered such a redesign. Process automation is commonly achieved by workflow or production automation systems, enterprise resource planning (ERP), or customer relationship management (CRM) applications. In contrast to a vendor-controlled scenario, each product deployment will require the execution of a project that analyzes, improves, and validates the business process changes that are enabled by the product.

The decision to develop a product offering for deployment to customer sites may result from an insight that the vendor's customer projects have significant potential for reuse. To exploit that potential, a company may engage in a stepwise productization process [ArWeBriFi10]. First, standard features are identified and managed based on proactive planning of the development. The reuse achieved with that first step can then be increased by further standardizing and packaging the features into a product. The customer projects use that product as a platform to develop customer-specific solutions. The last step of productization is the definition of a release train that offers a predictable agenda of updates to the product and minimizes the need for custom development.

As an alternative to product conceptualization in customer projects, a software product may be conceived by developing early versions of the product in the laboratory in intensive collaboration with a small set of selected pilot customers. Customer input may be obtained through interviews or workshops. To guide the prototyping, large companies tend to adopt technology readiness levels such as the

ones suggested by the European Commission.[3] TRL1-6 are usually research-oriented. Starting with TRL4, product management may take over the lead and bring the product to the market.

TRL 1 - Basic principles of the idea observed
TRL 2 - Technology concept formulated
TRL 3 - Experimental proof of concept
TRL 4 - Technology validated in the laboratory
TRL 5 - Technology validated in a relevant environment
TRL 6 - Technology demonstrated in a relevant environment
TRL 7 - System prototype demonstration in operational environment
TRL 8 - System complete and qualified
TRL 9 - Actual system proven in operational environment

Depending on the customer contact person, it may be sensible for the product manager to engage members of the sales and marketing staff and have them attend the events with the pilot customers. To avoid confusion and consequent customer dissatisfaction, a vendor must be careful to provide consistent, positive messages to the customer representatives in the various meetings and prototype demonstrations. For example, it is better to express a statement like "well, yes there are several problems now but those will be eliminated when you install the new code release when it is available next year" as "the product is excellent now, and we are continuing to invest to make it even better." If the communication is not under control, the enthusiasm of a lab developer for the next release can easily leave the impression that the current release has numerous problems and that any implementation ought to be delayed—the opposite of what sales would like the customer to take away from the sessions.

The deployment of a product into a customer's premises may surface several issues which might not be present in a vendor-controlled installation. Customers may have differing computation environments and be subject to a variety of regulations. Thus, the product may be subject to non-functional requirements that, if not satisfied, will prevent the customer from procuring the product. It is essential that a product manager knows these requirements and decides whether and how to support these environments. Quality requirements that are important for deployment of a system into a customer's premises are the following ones:

- Portability: a software should be able to run on different hardware and software platforms without any significant migration effort. This is desirable for software companies who want to sell to customers with heterogeneous computation environments as well as for customers who want to be independent from

[3]Horizon 2020, Technology Readiness Levels (TRL), http://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf, accessed on October 5, 2015.

platform vendors. Over time, portability has become easier to achieve by using modern languages and multi-platform development environments. Also, virtualization technologies ease deployment.

- Interoperability: Interoperability enables integration of a software product into an organization's system and application landscape. To achieve interoperability, a product vendor needs to understand or actively contribute to the standardization of interfaces or support data exchange with the systems that are most important for the business with the customers.
- Security and privacy: data that is business-critical for the customer, in particular data about the customer's customers, must be protected. Features for achieving security and privacy include identification, authentication, and authorization of users and confidentiality, non-repudiation, and integrity of data.
- Performance: a system must react within acceptable time to user requests and limit excessive use of memory, storage, and network resources. The definition of performance requirements based on realistic assumptions about volumes is one of the most difficult and most important topics before development is started. Hardly anything has such a strong influence on the architecture and the design of a software solution as the performance requirements. Such planning is especially difficult when computation and networking requirements differ widely among customers. Cloud technologies have advanced to the extent that the performance requirements planning problem is eased.
- Reliability: Reliability is the degree to which a software product can deliver correct results consistently. Reliability is important as otherwise security may be compromised or the trust of the users may be lost. A special reliability concern is robustness, the ability of a system to react adequately to erroneous inputs and user actions without error situations. Another reliability concern is availability. Many critical applications require an availability above 99%, i.e. allowing a downtime of roughly 8 h per month. The number of downtimes per timeframe may also be an important question. High availability may only be achieved by utilizing adequate hardware and software architecture elements like redundancy, hot standby, clustering of application components, etc. These must be implemented on the system and application levels. Fast restart in error situations, disaster recovery, and the use of backup computing centers may also be critical to some customers. In recent years, cloud technology has matured to the degree that common reliability and performance tactics can be easily implemented.

Kano Model: Must-Be Attributes (Dissatisfiers)

The third category of requirements that Kano suggests is called "dissatisfiers" or "must-be attributes" [KaSeTT84]. Dissatisfiers refer to missing attributes of a software product that have been considered state-of-the-art for an

(continued)

extended time. They represent dysfunctional characteristics of a system: characteristics that are missed by a stakeholder if they are not implemented. In contrast to the satisfiers, the dissatisfiers do not generate excitement or delight if implemented. The presence of a dissatisfier may lead to a reaction: a "no" to the procurement and use of the product.

The consideration of dissatisfiers is of importance in the icebreaker scenario, where the software product is deployed into the customer's premises. Here, the integration of the software product into the chain of tools that are used to support the customer's business process will not excite the customer. However, the absence of such integration will disappoint that customer. One of the common integration concerns in the context for many enterprise systems is for user account management. Lack of support for the relevant account management standards, e.g. for single sign-in, may lead to rejection of the product in a customer's environment even if the product satisfies the core stakeholder needs.

In a survey, a dissatisfier X can be identified with negative answers to a question like "How would you feel if our product did not have capability X?" paired with a neutral answer to a question like "How would you feel if you obtained a product with capabilities X?" These negatively formulated questions can be used to discover minimal product requirements.

Cruise Ship

The cruise ship situation is characterized by the presence of a product that has been installed by first customers and needs evolution. The product is deployed at customer sites by project staff, the vendor's own staff or partners', who use the appropriate product versions to develop and setup customer-specific solutions.

A factor that makes the cruise ship installation complicated is that customers use different product versions. Since these versions need to be supported, homogenization is an important concern of the product manager who needs to convince resisting customers to do updates and upgrades. Customers are often not interested in making updates as each update generates cost for adaptation, deployment, conformance checking, and training. Updates also introduce uncertainty about the proper functioning and stability of other customer systems. Microsoft, for example, struggles in convincing customers to adopt the most recent version of Windows, with the effect that versions need to be supported that are more than a decade old.

In the cruise ship scenario, the product manager is confronted with a stream of bug reports and feature requests that stem from a variety of sources. They may originate from customers' requests for proposals that are handled by sales staff, customer requirements that are handled by the customer projects, or incidents that are handled by customer support. The product manager receives requests from these company functions for product enhancements. Other sources of requests may be marketing, customers, sales channels, and development.

The *marketing* team can provide valuable inspiration for the design of a software product. Marketing is likely to supply information about what the product must be able to do so that it can be marketed successfully and which features could make the product more attractive. The way in which a product is presented on the internet, is packaged for sale, or advertised plays a major role as far as consumer products are concerned. The marketing staff can also be consulted for information regarding licensing terms and conditions, pricing, etc.

At any company the *sales and distribution* team has the most—and hopefully the best—customer contacts. This is reason enough for it to be an important source for gathering requirements. Due to their broad scope of responsibility, however, many sales people have trouble precisely defining requirements for a single product—unless the person concerned is a product sales specialist.

The requirements obtained from the sales and distribution staff are often vague or express the opinion of just one or a few customers. Yet, this information is important for the product manager, particularly as an "early warning system." The sales and distribution channel is often the first to be contacted by a customer who is unsatisfied with a product or product service. The product manager should respond appropriately before the customer switches to a comparable competing product. Meeting with the customer together with a development representative or organizing a requirements workshop often does wonders. The effect of this is that the customer is more satisfied—feeling that the vendor has taken his problems seriously—and that the sales specialist feels upbeat, having been able to define his fuzzy requirements more precisely together with the customer.

Caution must be exercised when developers meet customers, however. The very act of bringing a developer in to talk to a customer will allow the customer to infer an expectation that his requirement will be addressed even if the vendor has other intentions. It should be made clear that the meeting is to gather information and that there is no commitment to development. One must also guard against an excess of candor if it takes the form of "that is the silliest requirement I ever heard!"

With all the varied inputs, the product manager must develop a good instinct for identifying the real problems. Every member or head of the sales and distribution team—with minds focused on the next sale—tends to present his customer's requirements as the most important of all. This will often create a conflict for the product manager. On the one hand, he should always work in close collaboration with the sales unit, and on the other, he must prevent the sales and distribution staff from sidestepping him and submitting requirements directly to development.

Software products may be sold and distributed through *sales channels* like retailers, partners, value-added resellers (VARs) (see Sect. 3.11). They are often difficult for the product manager to "grasp." If the product is sold and distributed by third parties, the in-house sales department has limited direct customer contact and thus has difficulties in obtaining pertinent information for supporting product decisions. In this case, the retailers and partners are an important source of information for obtaining suggestions for new products or feedback on existing products. Such business partnerships work well if both parties benefit from them. A retailer or partner must be positive about the vendor's products, regardless of

whether he makes money out of the license or product consulting and implementation services. The reseller will be as keen to remove inhibitors to sales as the vendor and is a good source of such information. Retailer and partner events organized in cooperation with the partners' sales department are—like the user meetings discussed above—a good forum for the exchange of ideas.

Internal staff or external consultants may execute *customer projects*. They are a valuable source for gathering new requirements, especially for commercial software products that require a great deal of consultation, customization, and implementation services. External consultants are interesting for the product manager because they are independent, are often familiar with the products in various customer environments, and have usually already worked with similar competing products.

There are cases of *customers* developing a product "add-on" that may be interesting to include in the product offering. In such a case, the vendor needs to decide whether the "add-on" should remain customer-specific, should be integrated into the product, or should become part of an ecosystem strategy where the "add-on" is managed by a third party as an independent product on top of the vendor's product. It is advisable for the product manager to meet with the customer together with a development manager or a system architect. Another proven forum is a "customer advisory council" of carefully selected and representative users to review plans and comment on these plans before new functions are invested in. However, such a dialogue-oriented approach only works if the customers are representative of the target market and insightful about the underlying product technology.

Ideas about product enhancements and technological improvements generated by *product development* may constitute a further source of product requirements. These ideas often have a long-term perspective and tend to be justified with cost-saving arguments, while requests from the customer side tend to follow business development arguments. The product manager needs to balance these long and short-term needs and consider both lines of argumentation in the roadmapping and release planning decisions that follow product requirements engineering.

Also, the *support* team can be a helpful source of information for requirements management. The support department knows well the problems customers have—no other function receives as many customer complaints. It is worthwhile for the product manager to monitor and evaluate the problem database. The support function is also a good partner to collaborate with for customer satisfaction surveys. These surveys may not result in detailed requirements but can provide valuable pointers to product-related areas that need improvement.

The costs generated by support may be significant. Call center studies have shown that 80% of all calls concern the same 5–10 problems. In many cases, these problems involve installation, documentation errors, or problematic workflows. Improving the installation procedure, documentation, help functions or user interface can drastically reduce support costs as well as increase customer satisfaction.

The process of collecting, evaluating, and deciding about the stream of requests obtained from the many sources in the cruise ship situation is called triage

[Davis05]. Triage of requirements proceeds over a series of steps. First, candidate requirements are collected and evaluated based on strategic relevance for the product [GorsWohl06]. Selected requirements are then evaluated from the perspectives market value and development cost [KarlRyan97]. This evaluation and subsequent selection for implementation will be elaborated in the release planning chapter.

Kano Model: Indifferent Attributes (Non-requirements)
A fourth category of features useful to consider in the context of the Kano model [KaSeTT84] may be called "non-requirements" or "indifferent attributes." These features are those that the product manager should avoid. They refer to attributes of a software product that neither generates positive emotions when implemented nor negative emotions when not implemented. The indifference of the customers implies that development resources should not be wasted for implementing these requirements. A product that lacks these features is as attractive as one that has them implemented.
In the cruise ship scenario, it is common that the product manager is confronted with a stream of requests from various company functions. The filtering of indifferent attributes is important. A requirement that is perceived as critical for one customer may not be perceived as important by other customers. A requirement that may be considered exciting by some sales staff may not deliver the increase in product sales that was hoped for.
Features may be non-requirements for some customer segments and, at the same time, be important for other segments. When used in conjunction with market segmentation, the segments that react indifferently to potentially enhancing features may receive a high priority for early product releases. They may be offered product versions that have a limited scope, i.e. that do not include enhancements that must be gotten right for other segments.
In a survey, non-requirements can be identified with the same questions as satisfiers are identified: "how would you feel if you optained a product with capabilities X?" and "how would you feel if our product did not have capability X?" The non-requirements are those for which no strong positive or negative answers are received.

4.2.4 Internationalization

The decision to sell a product internationally may result in additional requirements compared to a purely domestic one. One common additional requirement is the support for multiple languages. If a product is first released in a local language and subsequently translated into another language to make the product available in another country, retrofitting the new languages is often time-consuming. National

language support enablement should be considered early in the product specifications and design. Such an approach requires the language-dependent parts of the product, the user interface, screen masks, messages, help functions, and online documentation, to be separated from the business logic.

When scaling a product to a global market, there are regulations, standards, certificates, and permits that need to be observed, met, or obtained for each specific country. Many customer environments and product domains are subject to such national regulations. A widely known example of a regulation that applies to customer environments is the American Sarbanes-Oxley act. It regulates financial practice and corporate governance and contains requirements that affect a company's business processes and the software products that support these processes. A widely known example of a regulation that applies to products is the medical device directive. It regulates products, including the software and hardware which affect the well-being or life of humans. In Europe, products that comply with the directive gain market access and obtain the CE mark.

A global product must consider cultural issues and local usage. A trivial example is the use of the period as a decimal separator and the comma as a thousand separator in the U.S., while in Europe and much of the world, the roles of those punctuation marks is reversed. Getting such points right may be crucial to the success of products in specific countries. Even in the case of purely functional requirements, countries have different priorities, often due to cultural differences. And there are certainly differences in the legal aspects like sales laws or warranty. As all these cases show, an internationally oriented product management that takes these aspects into consideration and prioritizes them properly is important to successful international development and sales.

4.2.5 Documentation of the Requirements

Requirements must be documented to be useful for building a shared understanding, planning the product, and coordinating the development. How requirements are documented does not matter [Fricker14]. Aggregated over more than 400 projects, no evidence could be found that requirements engineering success was influenced by decisions about templates for requirements phrases, box-and-line drawings, or even formal specifications. The way requirements are stored did not have any influence on requirements engineering success either. Specifications may be saved with documents, spreadsheets, Wikis, or requirements databases. What is important is the right methodology for building a strong business case and aligning the software product with the needs of the product stakeholders.

To be efficient, requirements engineering practice should comply with policies on the development side. These policies include standards for requirements specification, the toolchain in the company, and the needs of those who are working with the requirements.

The software development life cycle chosen by the development organization determines how to document requirements and ensure that the requirements are

implemented and tested appropriately. The extreme life cycle models are agile and Waterfall development. The Rational Unified Process represent the middle ground.

Agile development started as a development life cycle methodology and philosophy in the early 2000s. Agile development prefers interaction among individuals over a step-wise process, working software over comprehensive documentation, collaboration between product management and development over contract negotiation, and responding to change over following a plan.[4] The development proceeds as a series of rapid iterations that result in frequent demonstrations of a functioning system.

Development is planned and coordinated with backlogs rather than with specification documents. A backlog is a list of coarse-grained features that are refined into fine-grained requirements. When using Scrum as an agile process, the features are called Epics. Epics should be used for documenting a shared understanding of the meaning of the feature that the product manager had in mind. For achieving such a shared understanding, we found the template for documenting options useful (c.f. the table in Sect. 4.1.4, [FrGoBySc10]).

In Scrum, the requirements are written from a user perspective and follow the format of User Stories. A User Story has the following format [Cohn04, LuDaWeBr16]:

- Specification: a sentence that follows the structure "As a <user in the role X>, I want to <use the functionality Y of the software> so that <the benefit Z that shall result from using Y that I as a user am interested in>."
- Comments added as the user story is being discussed: any information important to correctly interpret the user story. The comments may include constraints that must be considered for implementation.
- Tests added as the definition of "done" are discussed: exemplary scenarios of how the system is used and test data defining how the software will react to user inputs. The tests may be further formalized with the Behavior-Driven Development approach [SolisWang11].

In an agile context, requirements are prioritized by sorting the backlog. The top entries are the most urgent ones and are implemented in the next development iteration. The lower entries are less urgent; some may never be implemented. A consequence of this approach is that no importance classes, such as mandatory and optional requirements, need to be defined. Instead, priorities are re-negotiated at the beginning of each development iteration.

Waterfall development emerged as a development life cycle model in the early 1970s [Royce70]. In Waterfall development, the upfront planning of the system development is important. The development is planned based on requirements specification documents that are created before the development starts. The

[4]Manifesto for Agile Software Development: http://www.agilemanifesto.org/

development then proceeds with two iterations, one for a prototype and a subsequent one for the operational system.

In a Waterfall development context, requirements management is a document-centered task. All requirements are combined into a document, the high-level specification of a product. This is the main working document of product requirements management. From this early requirements specification, the technical specification document is derived. The ISO/IEC/IEEE 29148:2011 and IEEE 830:1998 standards describe common templates for requirements documents. In a public tender context, the V-Model XT and Hermes processes provide good templates.

For Waterfall development to succeed, it is of key importance that the right requirements are defined, comprehensibly documented, evaluated, and categorized by their importance. Often, a two- or three-level prioritization scheme is used:

- Mandatory requirements: a requirement is mandatory if not implementing it would automatically result in potentially drastic consequences for the company, such as a loss of customers and revenue. Examples are contractual obligations with a customer or legal requirements.
- Optional requirements: these are the requirements that are less important. Of course, they can be further differentiated by urgency. Over time, an optional requirement can turn into a mandatory one.

To manage the risks of missed requirements, misunderstanding of underlying technology, and wrong estimates, the *Rational Unified Process* has been proposed. It suggests that each phase is executed with multiple iterations and by involving all development disciplines throughout the whole development process [Kruchten96].

When using the Rational Unified Process, the product manager is responsible for a vision document that development refines into a software requirements specification consisting of documents that describe use cases and supplementary requirements. The vision document refines the definition of the product strategy and is written for a technical audience that implements a specific product version. It contains the following information:

- Introduction: purpose, scope, definitions, and references.
- Positioning: descriptions of the business opportunity, the customer problem, and the positioning of the product.
- Stakeholders: user roles and profiles, competitors and the alternatives they offer, stakeholders external and internal to the vendor organization.
- Product Overview: interfaces to the product's environment, overview of product features, assumptions and dependencies, delivery and installation.
- Non-functional requirements:
  - Constraints, i.e business, project or design decisions taken in advance to ensure the solution fits business, managerial and contextual concerns,
  - Quality requirements, and
  - Documentation requirements.

Whether the development proceeds in an agile or Waterfall fashion, the product manager is expected to decide about quality requirements that complement the functional requirements. The decisions concern the prioritization of the types of qualities that are of importance and the definition of expected quality levels. The types of quality requirements are discussed in the speedboat and icebreaker subsections.

Finding the appropriate quality levels is difficult in practice, but may affect development cost and customer experience in dramatic ways [RegnBSO08]. Too much quality will force development to choose architectures that may be an order of magnitude more expensive than if less quality can be afforded. Too little quality will lead to bad quality experience for the customers and users that interact with the product. For critical quality attributes, customers will choose competitive products that deliver better quality. For determining the appropriate levels of quality, the product manager needs to understand the customers' priorities in terms of product quality attributes, competing products' respective performance from the customer perspective, and technical options and associated costs for building useful or competitive quality while avoiding quality excess. This work proceeds best in focused collaboration with Marketing and Development and may be supported with workshops where experiments are made to understand the effect of product quality on the user's quality of experience [FoFrFi14].

Many software organizations want traceability among requirements artefacts, in particular between customer, product and project requirements (see Sect. 4.2.6). That means the ability to follow the path from the original requirement to the software implementation and back. They often implement an integrated tool chain for managing these artefacts. An integrated tool chain will allow the product management organization to establish horizontal traceability from customer requirements to the product implementation and vertical traceability from strategic objectives to design decisions. The traceability connections are created by giving each artefact unique identifiers that could be referred to in the same way as web pages link to each other with URLs. Standardized notations and traceability give transparency about development plans and progress, thus allow rapidly introducing or reacting to changes. In regulated markets, such traceability may be a requirement for placing a product on the market. Also, in some environments the introduction of traceability has enabled systematic reuse of requirements, code, and tests to the extent that the time-to-market for a new product could be accelerated by 90% in a product line environment.

A Siemens unit, for example, has established the following traceability and toolchain [ClSeRBC07]:

- Stakeholder requests (documented with text documents) traced to features (managed in a requirements database).
- The features (managed in the requirements database) refined into system use cases (managed in a modeling tool).
- The use cases refined into requirements that describe concrete system capabilities (both managed in the modeling tool).

- Documentation of product releases (managed in a document versioning system) defining when the concrete system capabilities would be implemented (managed in the design modeling tool).

Complete toolchains are difficult to establish in practice because they require a restricted set of tools to be used in the development organization. These restrictions compete with usability for the people who need to interact with the tools and have their work effectively supported. Also, a continuous stream of tools is coming on the market that may change the way development is performed and implies the consequent changes in the toolchain.

Some organizations have introduced alternative forms of requirements documentation. Storyboards, whether hand-drawn or assembled with photographs, have been used to make descriptions of product use more intuitively understandable than when abstract specifications are used for such descriptions. Storyboards may be extended with videos showing the use of the product and how stakeholders react to such use [FricSFT16]. Such rich-media documentation is interesting in situations where the product organization needs to be convinced about the product context, where development work needs to be precise, and where there is a need for knowledge transfer and learning. In addition to being used for requirements documentation, photos and videos may be used to advertise products, to educate users about how to use the product, and to capture experiences of these users in support of product evolution.[5]

Whatever approach has been chosen to document the requirements, a product manager has also to decide about the depth of the requirements specification. This depth, or the level of detail, should be adapted to the maturity of the requirements understanding and the knowledge of the requirements receivers. A hint about a possible solution may be enough for an experienced subject expert that the product manager has collaborated with over long time. In contrast, junior developers may need to learn about many details before they can interpret the requirements correctly. For planning the appropriate level of detail, a simple test exists for testing requirements understanding: ask how the recipient will implement it. If you agree: fine. If not: change the understanding by adding detail or seek another team [Condon02]. Also, keep in mind that requirements tend to be vague and abstract early when product concepts are being explored and tested. Later when the product concepts have been chosen and solidified, the requirements must be detailed enough to facilitate project planning, product development, and testing.

4.2.6 Managing Requirements Engineering

Requirements are used to justify decisions among architectural options, to estimate and schedule work, to facilitate quality assurance, and to coordinate among

[5] https://youtu.be/CBsOiabbNIc shows an example of such a product video.

different products. Like all business processes that are cross-organizational and require the cooperation of different organizational units, the requirements management process is a big problem in a lot of companies.

The knowledge of who will implement a given requirement and when it will be implemented avoids redundant activities and allows planning of dependent work. An assigned and scheduled requirement can be reviewed, followed-up, and changed if necessary. The following paragraphs describe measurements that companies may take. The measurements make sense especially in the speedboat and cruise ship scenarios as the measurements are particularly useful for mature products where there is a desire to improve the requirements engineering process.

One essential measurement is *time-to-market*. This measurement tells how fast and flexible the product organization is to react to market changes and to implement strategic decisions. If time-to-market is too slow, capacity may need to be increased, product development focused, and the release frequency increased. The time-to-market measurements can be refined with measurements that reflect how fast and thoroughly requirements are worked on, e.g. the average time from the initiation of a requirement to the point of decision on implementation and to the availability of the product release that contains the implementation.

Another essential measurement is the *size of the requirements backlog*. This measurement tells how much work is pending. If based on a clear definition of the organization's requirements process steps, the measurement gives confidence that the flow of requirements through the process is not stopped at some point. When the backlogs are measured at different locations in the organization, the requirements flow becomes visible, and bottlenecks and overcapacity may be identified and eliminated. Ericsson, for example, applied such value stream analysis systematically to improve and balance the overall productivity of their product organization [PetWoh10].

A third measurement that is useful, particularly during requirements triage and when the scope of a development project is defined, is the *degree of confidence in requirements decisions*. Common confidence issues concern the linkage of requirements with business strategy, seeing the big picture of the offering, understanding the planned product's value, and knowledge of customer problems [KKTLD15].

The process owner who is measured on how well the end-to-end process works is a key success factor. The role of the process owner is necessary, even though it is not always pleasant. The management board needs to support him and give escalation rights to him. The following would constitute good assignment of process ownership.

- Customer requirements: business development, customer management.
- Product requirements: software product management.
- Project requirements: development.

Customer requirements management can be tightly connected with product requirements management. Project requirements management is closely connected

to the project management and development processes. The connection of project requirements management to customer and product requirements management is usually looser.

4.2.7 Summary and Conclusions

This chapter has outlined the contexts and ways that requirements are engineered and managed at the product level. Each of the product scenarios has its specific approaches to requirements engineering. The powerboat scenario features a lightweight, flexible experimentation process; the speedboat scales creativity supported by user feedback; the icebreaker follows a product development process that generalizes from individual customer projects; and the cruise ship has a continuous inflow of customer requirements that are triaged into product requirements.

Requirements documentation has to be adapted to the specific circumstances of the organization. Whether a formal standard is followed for requirements documentation may not matter for requirements engineering success. However, for efficient and smooth work, the requirements specification needs to be well integrated into the corporate work processes, culture, and toolchain. The product manager will specify and handle requirements in the context of agile development differently compared with a Waterfall development context.

For mature product companies, requirements represent an important input for process development. Time-to-market measurements, product backlog size, and confidence in the requirements will allow planning and balancing capacities and influence the way requirements decisions are taken.

4.3 Release Planning

4.3.1 Overview

Release planning refers to the management of the detailed contents and schedule of a forthcoming product release. In comparison to a roadmap, the release plan focuses on the short-term, a single release. Release plans are important because they define how the organization balances its capabilities for short-term goal achievement and long-term investment by adding, changing, and removing or re-focusing resources. The software product manager needs to make sure that the release plan is synchronized with the product roadmap and matches the organizational interests.

The release plan is used by the project manager to scope product development. Often, the development resources are not sufficient to implement all the identified requirements for timely delivery, implying a need for prioritization and debate between stakeholders to reach an agreement about trade-offs. Since such detailed planning leads to new insights, release planning may imply changes in the product roadmap that affect stakeholders other than just product development.

Release planning is done at regular intervals or at moments that are important for the product organization. Companies may synchronize development projects with a heartbeat to achieve consistency across multiple software systems. The heartbeat determines when release projects are started and when the new product releases are made available to customers. The frequency of the heartbeat varies, from one release per year to one release per few months to continuous releases.

Release planning should be supported with tools for managing requirements backlogs to handle the often large amount of detail. The tools, discussed in Sect. 4.5, give transparency about pending and completed work and may be as simple as a spreadsheet or as advanced as issue trackers that interoperate with project, test, and knowledge management tools. This transparency, reflected by changes in the backlog, gives an indication of remaining time-to-complete and of whether the right amount of development resources is available.

4.3.2 Concepts

The Release Concept

A software product evolves over time and thus exists in different releases. Releases have different size and visibility external to the product organization. According to the ISPMA glossary, releases that are offered to customers are called product releases, while releases that are only visible within the product organization are called pre-releases. Releases with large changes may be called versions, while releases with small changes are called increments. Figure 4.5 gives an overview of these terms.

A variety of labeling schemes for the released software is practiced by vendors. Depending on the type and amount of change, releases are called major, minor, update, and service or patch release. Other names are invented for marketing reasons. To use a widely-known software product as an example, Google numbered the latest release of the Android Gmail application available on February 3, 2016 "version 5.10.112808100" to reflect that the Gmail app was the update "112808100" of the tenth increment of the fifth version of the product. Another

Term	Definition
Release	An instance of software that is made available to stakeholders.
Pre-Release	A result of development activity that is testable, e.g. the result of a sprint in Scrum.
Product Release	An instance of the product that is delivered to customers, and maintained as part of product maintenance.
Major Release	The release contains significant new or changed functionality compared to other releases. It may be selected as a new version for marketing or business reasons.
Increment or Minor Release	A minor release that contains new or changed functionality compared to other releases.

Fig. 4.5 Release planning definitions

example, the Android operating system, was called version 6.0, "Marshmallow" when released by Google on October 5, 2015. As indicated by the increment "0", the release 6.0 was a major release. "Marshmallow" is a marketing name.

A product organization may offer multiple product releases at the same time. This may occur when customers cannot be convinced to install the latest version. Or a vendor may offer an older version at a considerably lower price to address price-sensitive markets. However, the support of multiple product releases in the market generates cost, diverts resources to maintenance, and thereby reduces the organization's capacity for developing new releases. For that reason, product organizations typically try to keep the number of concurrently maintained releases small.

The Release Planning Process

Releases are planned by selecting requirements for implementation. A variety of sources offer requirements from the perspectives of market needs and technological opportunities. Feedback obtained from customers and users is the most direct source of input on market needs. Marketing and sales may offer insights into trends and what customers are interested in buying. Support is exposed to the limitations of a product and knows about bug reports and feature requests. Other sources are partners, professional services, competitors, market research, and executive management. Electronic sources like app stores, social networks, customer relationship management databases, and product analytics offer a wealth of information about the use of products, customer needs, and opportunities for product improvement. Development, operations, and research may offer insights into technological opportunities.

To achieve short time-to-market, a first version of a new product should be small, yet still viable for the first customers. Later releases enhance that minimally viable scope and make the product attractive for customer segments that were not addressed with the first version. When a product achieves maturity, release decisions aim at keeping the product competitive and interoperable. At the end of the life of a product, release plans may contain features that help customers to migrate to replacement products. Section 4.4 describes how the product life cycle affects product planning.

Figure 4.6 shows how the inputs obtained from the market and technology perspectives are used in a step-wise refinement and selection process for deciding about a release for a mature product. Candidate requirements are received and specified, are evaluated in terms of value and cost. Release plans are created by selecting, or making a triage, among the requirements based on the budget that is made available to the release projects. The selection of requirements to be implemented is called the scope of the release. As the analysis and development unfold in the release projects, release plans are reviewed and adjusted until a dependable commitment can be given for what the exact scope is that is about to be released. Each instance of the release plan that is communicated is called a baseline.

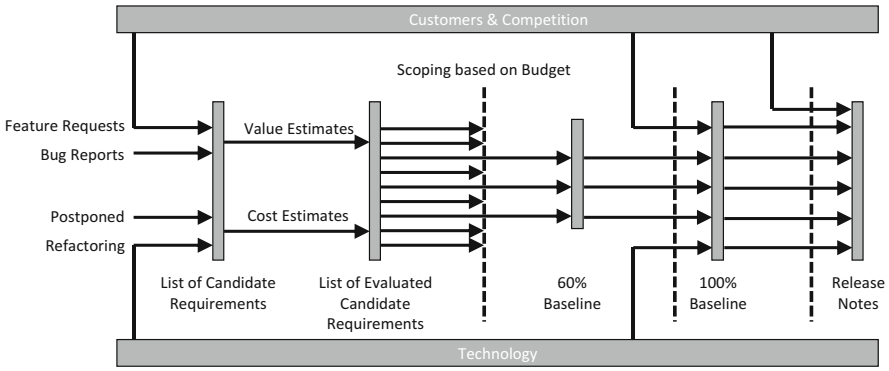

Fig. 4.6 Requirements flow and triage for a mature product, adapted from [Davis05]. *Vertical gray bars* requirements baselines. *Vertical dashed lines* triage decisions

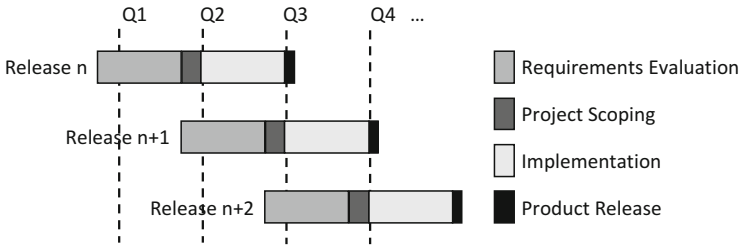

Fig. 4.7 Parallelization of release development projects with a release train, based on [Reg HNBH01]: shown is an extract with a series of 3 release projects over four quarters (Q)

Release Trains, Staging, and Product Variants

To minimize the time-to-market and utilize specialist capacity efficiently, software organizations work with parallel projects. A heartbeat is used to define a schedule of releases that is called a release train. Figure 4.7 shows such a schedule. The release train below shows clearly how this development plan allows compact utilization of requirements engineering and development capacity. Management and other company functions are involved on a regular schedule for project scoping and product release. Notice that the term "release train" is a central term in SAFe [Leffing16], but defined differently. Here an "agile release train" is the team of agile teams working together.

In many situations, the release development work is structured with a staging process that aims at delivering software product releases that are attractive and mature. When the staging process is followed, the implementation of requirements must meet a series of quality gates. At each quality gate, a pre-release is created, labeled according to the achieved maturity, and made available to those who are

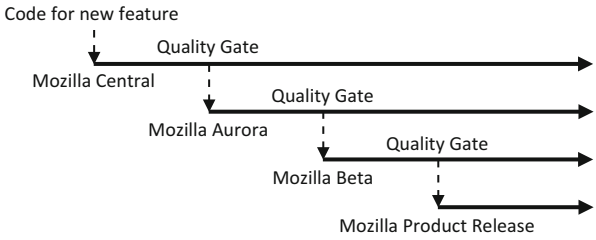

Fig. 4.8 Release of features in the staged Mozilla environment

maturing the software towards the next stage. Mozilla, for example, requires the features of the Firefox Browser to be matured in a four-staged environment.[6] Figure 4.8 illustrates this Mozilla staging process.

The Mozilla staging process includes the following releases:

- The nightly Central release allows experimental features to be exposed to the developer community and tried out.
- The subsequent Aurora release allows developers and early adopters to work with the features.
- The subsequent Beta release allows a release candidate of Firefox to be tested before it becomes the official product release.
- Finally, Firefox Release is the official product release of Firefox.

The code for a feature is in a Mozilla stage for at least 6 weeks, implying a minimum time-to-market duration of 18 weeks. The work for a feature may have much longer lead time, however. A feature is only branched into the next more mature stage if it has met the criteria of the quality gate that guards the stage. The staging allows the community to work on multiple Firefox releases at the same time while assuring high quality of the published product releases.

Release planning becomes a complex multi-dimensional problem when product families, lines, or platforms are involved. Versioning across product variants adds complexity in addition to the time-oriented versioning described above. A product manager is confronted with horizontal versioning if e.g. the same product is offered in different industry-specific incarnations or on multiple platforms. Vertical versioning signifies the offering of product variants with differing functionality and price points. When a product portfolio comprises a bundle of separate products with dependencies on each other and individual release plans, release planning and execution for the bundled product can become a nightmare. Hence, the product manager must do all that can be done to simplify the portfolio of variants and the release planning process.

[6] https://wiki.mozilla.org/Release_Management/Release_Process (accessed in February 2016).

Small vs. Large Releases

Whether to choose smaller releases with higher frequency or bigger releases with lower frequency are often a discussion point in a development effort. Big releases can be implemented more efficiently than small releases and sometimes are unavoidable due to requirements whose implementation would otherwise span multiple releases. Products with hardware elements tend to be developed with big releases. In any case, the release cycle should be short enough in order to avoid changes in the target market.

Small releases require less effort, can be implemented faster, and offer greater flexibility to account for changes in customer situations and changes in the competitive landscape. However, due to the high fixed cost per release for product and project management, regression testing, packaging, distribution, and sales and marketing, small releases are costlier unless significant parts of the process can be automated. Also, the larger the number of variants that are in use the higher the support complexity and cost.

The extreme case is the continuous delivery approach in which software may be released, e.g. in the extreme case of Amazon, every few seconds.[7] Software quality is managed by promoting changed software through testing into production in a semi-automated manner. The advantages of continuous delivery are accelerated time-to-market, rapid feedback from users on new requirements, and ultimately improved productivity and product quality with the consequent customer satisfaction. The biggest challenges are the resolution of conflicts among stakeholders, a software architecture and runtime environment that allow micro-changes during runtime, integration with slower processes of the organization, and the setup of a tool suite that enables continuous delivery.

A lot of software companies combine frequent small releases with infrequent big releases. A common industry approach, e.g. in the context of maintenance, is the release of critical fixes as they are developed followed by an entire maintenance release. The maintenance release combines the previous fixes and may add changes that can only be offered in conjunction with all the critical fixes. An example is Windows XP, then SP1, then SP2, and then SP3 with critical fixes all along the way. The product manager must keep an eye on upwards and downwards compatibility between releases and versions as described in Chap. 2.

4.3.3 Release Planning Methods

Various methods exist to answer the following questions: when and what should be released? The release planning decisions that drive the answers must balance opposing forces. On one side, the selected requirements need to satisfy business objectives and real customer needs while leading to a recognizable advantage over the competition. Such value consideration needs trade-off between technology push

[7] https://www.thoughtworks.com/insights/blog/case-continuous-delivery

and market pull, i.e. between technology-driven features and customer requirements. The organization's strategy may guide this trade-off and determine to what degree the company is reactive to its markets by giving priority to customer needs or pursues a proactive innovation strategy by pushing new technologies to the markets. On the other side, the selected requirements need to take into account the capabilities and capacity of the product organization, while being compliant with time and budget constraints and architectural considerations. Other influencing factors can be customer commitments and sales and marketing activities like a product presentation at a trade fair.

Deciding About a Release Plan

Academics have done interesting work on this subject by considering release planning as an optimization problem. Major factors considered in such release plan optimization are the value and effort estimated for individual requirements. Other factors are marketing themes attached to new releases or versions, dependencies between requirements and other products, customer commitments, and risk. Formal optimization approaches easily deal with value, effort, and dependencies. Other factors are more difficult to account for. Consequently, the optimization of release plans has been called the "Art and Science of Software Release Management" [RuheSal05]. The difficulty of formalizing a release planning problem is a reason for why many product managers do not use mathematical optimization yet.

In practice, release planning involves negotiations and setting priorities to resolve conflicts between stakeholders about release objectives and contents. An approved release plan is one that all stakeholders agree to, and conflicts are inevitable. Typically, a multi-stakeholder agreement is an iterative decision-making process of requirements elicitation, concept validation, and decision-making. Also, the software development methodology will affect the release planning. Agile and lean approaches allow frequent consideration and re-consideration of requirements' priorities, thus support a trial-and-error learning process of what is acceptable to the stakeholders. In contrast, Waterfall development freezes project scope and formalize change management early. Hence, this

Objective: agree with stakeholders on the scope of the next release project.
Steps:

1. Understand the release objectives and themes, the product and organizational environment, the parameters of the development, and the product's current life cycle phase that influence release decisions.
2. Prepare the release planning support and schedule.
3. Generate, evaluate, and negotiate the release plan, and make decisions.
4. Analyze, reflect, and package the release planning outcome and make the experience available to others to support organizational learning.

Fig. 4.9 Generic approach for deciding about a release plan

latter development methodology requires more upfront thinking and, consequently, more firm agreements.

Figure 4.9 shows a generic decision-making process for a software product release. Even though presented in a linear fashion, the process is highly iterative because the value of a release is hard to judge [Carlshamre02]. The evaluation criteria can often not be defined in advance. Judgments are highly subjective, and the grouping of requirements and features affects the value and cost of their implementation. As a consequence, it is difficult to assess whether a release content is good enough.

Release planning work requires significant preparation. In relation to step 1, a solid understanding of the product and organizational environment helps the product manager in the execution of the release planning process. Knowledge of the stakeholders, requirements, preferences, constraints, and cost constraints allows the product manager to conceive tactical options for alternative release plans and streamlines the release planning process. The preparation concludes in step 2 by selecting analysis and visualization approaches and defining a schedule of stakeholder workshops.

The step 3 is iterative and involves discussions between the stakeholders concerned with the release. The critical stakeholders are product management, engineering, marketing, and executive management. To facilitate scoping of the release consensus is necessary among the important stakeholders concerning the evaluation and prioritization of requirements and skillful visualization of the results.

Release planning work is concluded by analyzing the process and the outcome that has been achieved. This analysis aims at identifying opportunities for improving release planning. It involves investigation of the stakeholders' satisfaction with how release planning was performed and the appropriateness and stability of the release planning decisions. These outcomes should flow into the preparation of subsequent iterations of release planning thus avoiding unnecessary problems proactively while achieving an efficient and flexible process.

Evaluation Criteria

The evaluation, prioritization, and packaging of requirements into releases require creativity and experience. The ideal release, of course, is well-balanced, perfectly timed, and contains the most significant requirements that are of value for the customers, are required to fulfill the organization's strategic initiatives, and satisfy the needs for quality improvements. Obviously the release must be accepted by all product stakeholders. This ideal will rarely be achieved in practice. Usually focus areas must be defined for a release, e.g. making it a release for a new customer segment, a quality release, or a migration release that supports a new database system.

There are always more requirements than can be implemented with the given resources and budget. Also, time restrictions need to be considered, e.g. to meet the deadlines of trade fairs or answer announcements of competitive products. In some companies, a business evaluation of the individual features is to be done. That

Prioritization criteria for value
- Priority of stakeholders: prioritize by the beneficiaries of the requirements.
- Priority of customers and users: prioritize the requirements of prioritized customers.
- Competitors: prioritize requirements that allow equalizing or winning over competitors.
- Urgency: prioritize requirements by considering the product's release date.

Prioritization criteria for cost
- Effort: prioritize requirements that require little effort for implementation.
- Development cost/benefit: prioritize requirements with high expected benefit and low cost.
- Staff: prioritize requirements for which the right knowledge and skills are available.
- System operation: prioritize requirements based on the expected cost of system operation.
- Availability of support: prioritize requirements for which technical support and training can be provided.

Prioritization criteria related to the development of the technical solution
- Complexity: prioritize requirements that impede architectural degradation of the systems.
- Evolution: prioritize requirements that bring flexibility for system evolution.
- Dependencies: group requirements that should be implemented together and prioritize them by technical and marketing-related dependencies.
- Volatility: prioritize requirements that should be stabilized or isolated.

Fig. 4.10 Common prioritization criteria based on [WohAur05]

means that a business case is calculated that considers the cost side, in cooperation with development for the estimation of effort, and the value or benefits that an implementation of the requirements would bring. Other aspects to be considered are dependencies and grouping of requirements so that a combined implementation means less effort or less cost and can be explained to customers and stakeholders. Also, the resolution of conflicts between marketing and development becomes an integral part of requirements prioritization negotiations. Based on this evaluation, the product manager needs to decide what is to be done given the resources, budget, and time restraints.

Various criteria are used for requirements evaluation from the perspectives of the business, management, and system [WohAur05]. Business criteria are based on the product strategy. They include marketing concerns, e.g. release themes and dates, the competitive situation, stakeholder priorities, and requirements volatility. Management criteria include value, cost, and risk of the development, the needs for resources and competencies, the delivery date and calendar time in relation to the roadmap, and the availability of support for education and training. System criteria include system impact, complexity, requirements dependencies, evolution, and maintenance. Requirements dependencies increase the complexity of release planning and require additional analysis to be performed as the removal of any prerequisite requirement for the release affects the dependent requirements. Also, care must be taken on requirements that span teams or delivery cycles. Figure 4.10 gives an overview of the most common prioritization criteria.

The products' life cycle affects the selection of release planning criteria. In the early phases of the product life cycle, a high degree of uncertainty characterizes release planning. The product is not yet completely shaped, and the product use cases are evolving. Release planning focuses therefore on the most important

requirements that are needed for onboarding of the next customers and learning by the product organization. To ease change and evolution, a minimum viable product (MVP) approach is pursued (see Sect. 3.5.3).

In the growth phase, releases are planned that extend the MVP with features for satisfying interesting market segments and countering competitive threats. Also, interoperability is a key concern as it allows a product to be integrated into the customers' environments and increases the customers' difficulty of switching to alternatives. Because of the many additions, the product evolves rapidly, and complexity becomes an issue.

In the maturity phase, release planning no longer focuses primarily on the addition of new functions. Instead, requirements are prioritized that support new environments, improve user experience, and increase performance and reliability. Since the product architecture and constraints are well known, requirements can be estimated more precisely than in earlier phases, which leads to improved release planning outcomes.

Towards the end of the life of a product, requirements are prioritized that help to retain customers with minimal investment and help them switch to a replacement product that has hopefully been added to the company's portfolio. Large and disruptive improvements are directed towards the replacement product to avoid stakeholder inertia that hinder product replacement.

Prioritization Techniques

Several requirements prioritization techniques are commonly used. Techniques that are simple and can be employed ad-hoc are used to evaluate requirements in workshops with the concerned stakeholders. The more advanced techniques, which are based on mathematical optimization, require tool support and are used to generate and analyze options for release plans off-line in the preparation of decision-making workshops.

Figure 4.11 gives an overview of the techniques that can easily be employed in a workshop setting. In such a workshop, a moderator prioritizes requirements together with stakeholders according to the chosen evaluation criteria. To ensure the credibility of the results, the workshop will involve the stakeholders that have the best possible knowledge about the concerned requirements and estimates. Market representatives should evaluate requirements value, and developers requirements cost. The prioritization is repeated for each prioritization criterion. When stakeholders are confronted with many requirements, the requirements are aggregated, e.g. into features, before prioritization.

Especially in agile settings, the Planning Poker technique is used for estimating effort. It can be shown that the technique is efficient and leads to reliable estimates [KarTRBW06]. Also, the technique is interesting because the differences between estimates can be used as an indicator for uncertainty, and disagreement can be resolved through a dialogue among the participants. These two aspects let risk be exposed and controlled, thus let the participants develop trust in the prioritization results. In addition to effort estimation, the technique can be used for evaluating requirements according to any other criterion as well.

Top-10
1. Prepare: the moderator introduces the requirements and the prioritization criteria.
2. Evaluate: each participant selects the ten most relevant items privately, without sharing the results with any other participant.
3. Aggregate: the moderator counts the number of votes for each requirement.

Success enhancer: in step 2, 10%-30% of the requirements should be selected.

Numerical Assignment (Grouping)
1. Prepare: the moderator introduces the requirements and the prioritization criteria.
2. Evaluate: each participant privately partitions the requirements into three groups, for example, into "critical" (weight 9), "important" (weight 3), and "optional" (weight 1).
3. Aggregate: the moderator calculates the weighted sum of votes for each requirement.

Success enhancer: in step 2, each group should have approximately the same size.

Ranking (Sorting)
1. Prepare: the moderator introduces the requirements and the prioritization criteria.
2. Evaluate: each participant privately sorts the requirements so that the best-ranking requirements is at the top and the worst-ranking at the bottomof the list.
3. Aggregate: the moderator calculates the median rank (more robust) or average rank (enables many statistical tests) of each requirement.

Success enhancer: allow participants to sort visually, e.g. with a spreadsheet.

100 Dollar (Cumulative Voting)
1. Prepare: the moderator introduces the requirements and the prioritization criteria.
2. Evaluate: each participant privately invests a total of 100 points (or dollars or euros) into the set of requirements, proportional to the perceived worth according to the criteria.
3. Aggregate: the moderator sums the number of invested points for each requirement.

Success enhancer: vary the number of points to be invested as necessary.

Planning Poker for Effort Estimation (Consensus Seeking)
1. Prepare: the moderator introduces the requirements and the procedure.
2. Evaluate: each participant privately estimates the effort for implementing the requirement.
3. Aggregate: each participant shares his estimate and, if it is the highest or lowest estimate, provides a justification for it. The group discusses the estimates and justifications.
4. Seek Consensus: go to step 2 if no consensus has been reached. If the differences between the most optimistic and pessimistic estimates are small, select an estimate in the middle.
5. Repeat the Planning Poker for all requirements that are to be evaluated.

Success enhancer: make estimates based on a Fibonacci-like series of numbers. A commonly used series is 0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 100, ?, and Coffee. "?" reflects the inability to estimate. "Coffee" reflects a need to make a break, e.g. for private discussions.

Fig. 4.11 Prioritization techniques for workshop settings, in the order of increasing complexity

When selecting requirements for implementation, multiple criteria need to be combined. Wiegers suggests to combine the benefits from need satisfaction, the penalty of need dissatisfaction, the cost of requirements implementation, and risk into an aggregate score [Wiegers03]. The requirements selection problem then becomes the problem of selecting the best-scoring requirements. Also, Ruhe and his colleagues consider requirements selection as a multi-criteria optimization problem. A further approach is the Analytical Hierarchy Process that allows defining weights for criteria in addition to evaluating the requirements with the criteria. While providing fine-grained rankings and supporting sensitivity analyses, these approaches suffer from a dependency on tools and from giving the

Quality Performance (QUPER) Approach
1. Prepare: define the type of quality requirement to be defined.
2. Evaluate benefit: estimate the breakpoints of useless, useful, competitive, and excessive quality from a user perspective.
3. Evaluate cost: estimate barriers for quality levels that require a significant change of the technical approach.
4. Analyze market: estimate the product's current quality and the current and expected quality of the competing products.
5. Propose releases: estimate quality targets for coming releases that reflect technical feasibility and the product's competitive strategy.
6. Repeat by extending the analysis with other important quality characteristics.

Success enhancer: measure the users' perceived benefit with a Mean Opinion Score "MOS." The MOS is on a scale from 1-5 and can be determined experimentally by letting users use the concerned features and asking them for feedback [FoFrFi14].

Fig. 4.12 QUPER technique for planning quality levels, based on [RegnBSO08]

impressions of being black boxes. They deliver results in ways that are not understood by the stakeholders. Also, the estimates of value and cost are often questionable even if sophisticated estimation approaches are used. Accordingly, the simpler techniques are much more frequently used.

Releases are often planned based on the Pareto principle. Pareto analysis is a technique to match the most important requirements with the given delivery capacity. The technique is based on the 80:20 principle that says that 20% of the work will generate 80% of the value. When following this approach, the top 20% of the analyzed requirements are selected. Davis recommends starting development with a requirements baseline that represents 60% of the capacity available in development [Davis05].

While the so far introduced techniques work well for functional requirements, they are of limited use for quality requirements. The definition of quality requirements, such as performance, usability, and reliability, is often a problem of defining appropriate levels of quality, rather than defining whether to add or remove a quality requirement. Regnell and his colleagues proposed the quality performance QUPER approach that determines quality levels based on competitive analyses and architectural barriers [RegnBSO08]. QUPER is a simplified form of the more elaborated Quality Function Deployment approach that is established for hardware products. Figure 4.12 gives an overview of the QUPER technique that is performed by a team of analysts that are competent both in technology and the market.

Visualizing and Agreeing on Prioritization Results

Visualization makes the requirements evaluation results accessible for the decision makers. It is a tactical tool that can be employed during the release negotiations. Visualization can be used to underline how collected evidence supports or speaks against the arguments that are used when deciding about a proposed release plan.

Evaluation results may be visualized in text or graphical format. A widely-used approach is to use sorted requirements lists or tables that indicate selection and

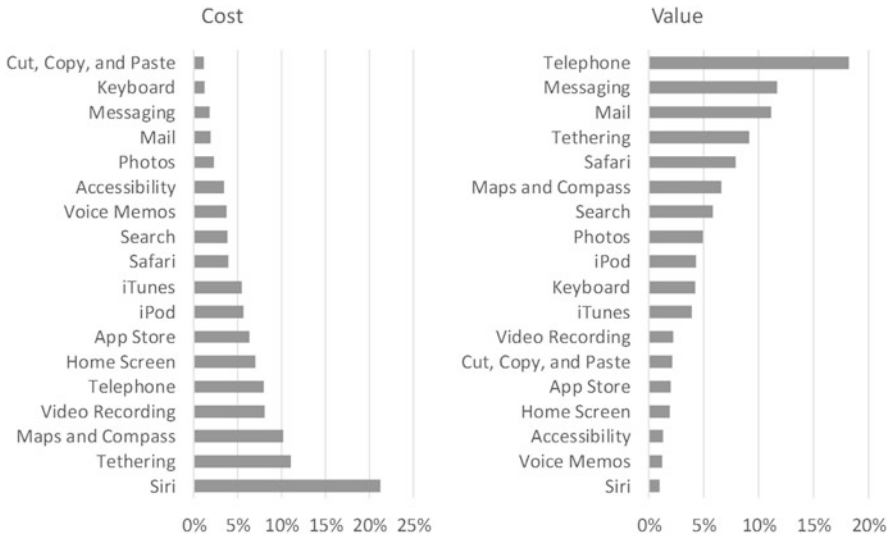

Fig. 4.13 Visualization of evaluation results for smartphone features with bar charts

discarding of requirements. In an agile context, the sorted requirements lists are called requirements backlogs. In a product line context, often tables are used that show the matching of requirements to product variants. Both visualization approaches may be enhanced with information about ratings and rankings, by adding columns with the respective information. When used for supporting decisions, the text-based visualizations offer insights both into the evaluation results and the detailed data.

When estimates are on a proportional scale, graphical representations are attractive. Their advantage is that they can express patterns and relationships better than text-based formats. A simple approach is the use of bar charts, illustrated in Fig. 4.13 that shows evaluation results of smartphone features. Bar charts can also be visualized as stacked bars, e.g. to indicate the contribution of individual stakeholders to the total estimate. Graphical formats may also be used to indicate the spread of estimates. For example, frequency distribution charts show the spread of values [RegHNBH01] and box-plots aggregate information about these distributions [McGiTuLa78]. These are common tools used by statisticians to give an overview of estimates.

When relationships between estimates are of interest, scatter plots become a useful visualization tool. In comparison to one-dimensional graphical representations, scatter plots make interesting data points, e.g. outliers, and clusters of data better visible. In release planning, scatter plots may be used to show the cost/value relationship of requirements. Figure 4.14 shows a cost/value diagram of the evaluation results shown in Fig. 4.13. Each point represents a feature: the x-axis the relative cost and the y-axis the relative value of a feature.

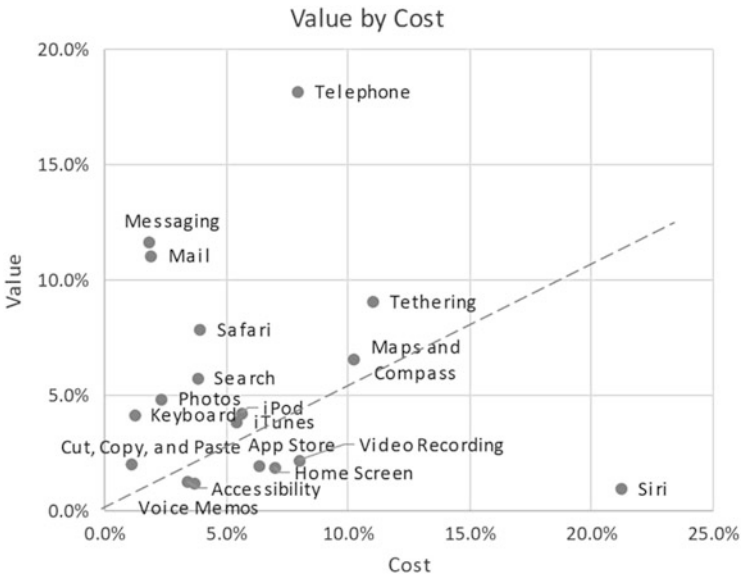

Fig. 4.14 Visualization of evaluation results as a cost/value scatter plot. *Dashed line* selection

Figure 4.14 illustrates the use of a cost/value diagram for selection of features with a dashed line. If all features above the line are chosen for implementation, the selection will imply the creation of 83% value for 46% cost by a development project in comparison to the possible total. Such selection of features leads to a large return on investment. Similar diagrams and similar analyses can be made for other combinations of evaluation criteria, for example, the criterion of risk.

More advanced charts have been proposed by Regnell and colleagues who suggest the use of disagreement charts, satisfaction charts, and influence charts [RegHNBH01]. The disagreement chart shows how much stakeholders agree or disagree on the priority of each feature. It allows identification of problematic features. The satisfaction chart shows how much each stakeholder's priorities agree with the overall priorities. The influence chart, finally, shows how much weight is given to each stakeholder. We refer to Regnell's publications for the details of how to construct these diagrams. What is important is that the product manager know who disagrees, who is satisfied, and who has what kind of influence. That knowledge allows acting by reviewing and influencing the decision-making.

To get to an agreement when multiple stakeholders are involved, the so far introduced analytical approaches are not enough. Reaching an agreement depends on the skillful use of the evaluation results, the development of convincing arguments, and the willingness of all involved to adjust their position to reach an agreement. This readiness for a successful dialogue can only be reached with trusted relationships among the involved parties, by having prepared alternative

proposals for release plans, and with an in-depth understanding of the impact of these plans for the stakeholders.

It is also important to know that a product manager does not need to obtain the support of all stakeholders [FricGru08]. An alliance of stakeholders suffices if the allied stakeholders have enough power to make the decision. If an agreement is not possible, the product manager may propose a middle way between the stakeholders' positions, a consensus. If conflicts persevere, the product manager may initiate an escalation of the decision-making to an authority with power stronger than the involved parties, for example to the head of the business unit or company to which the product belongs.

4.3.4 The Release Plan

The results from release planning are documented in a release plan. The release plan defines the core product in terms of functionality, quality, and constraints. The functionality reflects the services that the product offers to its users, the quality how well these services are offered. Section 4.2 on requirements engineering offers more details about the types of requirements. In addition to the specification of the software, the documentation for a software product release may contain the definition of the whole product, including non-software parts like hardware, documentation, and services such as consulting, pre-sales, post-sales services, and support.

There is no standard format for a release plan. The development and release process will guide how a release plan is documented. The simplest versions of release plans may be encountered in agile development projects. The most elaborated versions in a product line or systems-of-systems environments with staged releases of the product under development. The product manager needs to understand this product development context and adapt the format and contents of the release plan.

Minimal elements of a release plan include:

- A definition of the software version,
- The release date,
- Resource assumptions, and
- Selected requirements.

Many release plans include additional information:

- The intent-summarizing theme of the release,
- Supported stakeholders, and
- Traceability to the sources of the selected requirements.

Release plan may be even further enhanced with the following information:

- The customers or beneficiaries that are targeted with the software release,

- The budget, capacity, and staffing of the development project,
- Assumptions, dependencies, risks, and other issues, and
- Definition of product variants with allocation of the requirements.

Traditionally release planning practice documents the scope of a release in a succession of requirements documents. The marketing requirements document (MRD) documents the market opportunities that are to be addressed with the release. It is the first document to capture the list of evaluated candidate requirements. The product requirements document (PRD) consolidates relevant customer requirements into a coherent vision of the planned product release that considers the budget that is available for the release project. The technical requirements document (TRD) is a low-level specification of functional requirements, quality requirements, and constraints for the software to be developed. It defines the 100% scope of the software product.

Modern approaches for requirements specification are based on the idea of backlogs used to manage software development. These approaches are frequently used in the powerboat and speedboat constellations. Rather than writing requirements documents, Wikis and issue tracking tools are used to define plans and monitor progress. The plan that was historically defined by the documentation of requirements in a document is here defined by allocating the requirements to a release. The progress that was historically visible by the progressing set of documents is here managed by defining and following the life cycle of individual requirements.

During the release project, the product grows and matures. Upon conclusion of the release project, the documentation of the release must reflect any updates to the original documentation of the product. At this stage, the release notes replace the earlier release plan with a description of the implementation requirements together with an overview of known issues and bugs for the release. The product documentation includes, also, the technical artifacts, including architecture, development environment, code, test environment, tests, and test reports.

4.3.5 Summary and Conclusions

This chapter has given an overview of the concepts, process, and techniques for planning software product releases and documenting a release plan. Release planning has been presented as an evaluation and selection process, in which requirements from market pull and technology push undergo triage and are allocated to release projects that are lined up in a release train. To decide on the scope of a release, the product manager seeks agreement and support from important stakeholders with proposed plans that are developed by prioritizing the requirements according to criteria like value, cost, and risk. The chapter described the generic release planning process and gave an overview of the many techniques that are available. The chapter has concluded with suggestions for documenting the

release plan, both with backlogs for the modern agile environment and as a series of documents in more traditional environments.

4.4 Product Life Cycle Management

4.4.1 Overview

Product management is responsible for a product throughout its entire life cycle. The product manager conceives the product, introduces the product to the market, facilitates its growth, revitalizes it, and withdraws it at the end of its life. Each product life cycle phase affects the product manager's focus and how he does product planning. In the early stages, experimentation with the product concept and with new features is important. In the later stages, maintenance and replacement get center stage.

Not only the product, but also the category of the product undergoes its own life cycle. In the early stages of the product category, the technology adoption life cycle applies. Enthusiasts and visionaries become excited about the new capabilities and try to build new business with the technology. This phase is followed by the chasm where success in market niches determines the survival of the technology. A successful product category will continue to grow one niche market after the other until a widespread appeal has surfaced and a market has been created. That market grows, matures, and eventually declines. This category life cycle affects product planning as well. Each product category phase has a predominant type of new customer who is receptive to the product category and needs to be addressed if a product is to be sold successfully.

This chapter introduces the concepts of product life cycle management. The reader obtains an understanding of the three relevant life cycle and their stages, and recommendations about common tactics to implement product planning. Our primary focus will be on new products for immature markets and product evolution for mature markets.

4.4.2 The Product and Product Category Life Cycles

Product management must have a solid understanding of the various phases of a product and the product category to which it belongs. This understanding is needed to develop strategies and actions that suit the product's situation. Also, the product manager needs to ensure that the right knowledge for building and maintaining the product is available in the product organization. Finally, a product manager must analyze how well the product is performing by monitoring product profitability, actual versus planned revenue, customer satisfaction, and market share (see Sect. 3.13). If necessary, corrective actions must be initiated, and the concerned product team must be supported and given the competencies necessary to implement these actions.

Phase	Business Aim	Focus Areas
Conception and creation	Investment	Innovate, position product
Market Introduction		Launch product, grow market share
Growth		Grow market share, extend functionality
Maturity	Cash Cow	Revitalize product, service product
Decline		Retain customers
Withdrawal		Retain customers, reduce cost

Fig. 4.15 Life cycle model for a software product

Life Cycle Model for a Software Product

The importance of the product life cycle for a company is evident when looking at the product portfolio and how that portfolio evolves over time (see Sect. 5.2). Portfolio evolution may be measured by recording the number of new product sales or active licenses. The recording of new product sales is common for products in an early life cycle phase where the growth of the business is of key interest. The recording of active licenses is common for products in a late life cycle phase because that record allows knowing who is still using the product.

The growth and decline of business volume for a product version over time reveals the different stages in its life cycle. We see a product moving through six life cycle phases: the product is conceived and created, is being introduced to the market, is being grown, matures, and declines until it is withdrawn. Each phase of the life cycle has its individual characteristics, business aims, and focus areas for the product manager's attention. Figure 4.15 gives an overview.

A product organization pursues different business aims depending on the product life cycle phase. The first three life cycle phases of a product are investment phases. Investments in the product are necessary to develop, test, and market the product. Products in later phases serve as cash cows and generate significant revenue with relatively little investment. The resulting profit can be used for investing in other promising products in the portfolio. Often, the product manager will minimize the need for additional investments, maximize the revenue generated by the product, and achieve break-even as rapidly as possible. Agile approaches combined with a minimal viable product strategy and rapidly released product versions may yield quick return on investment by generating positive cash flow much earlier than a Waterfall-oriented approach [DenCle04].

Figure 4.15 shows the focus areas for each product life cycle phase. During *conception and creation*, the product manager assesses market opportunities for product ideas, aligns a winning idea with company strategy and positions the product, and develops and tests the product [SongMon98]. The *market introduction* includes the validation of the product with customers, testing of the marketing and advertising programs, and coordination, implementation, and monitoring of the new product launch. During *growth*, the product manager markets, extends, and evolves the product to win new customers, gain market share, and fight against competitors who do the same [Moore14]. With the *maturity* phase, the business aim

changes, and the product assumes the role of a cash cow. The product manager starts limiting the investments in the product and evolves it only as much as needed [RajlBenn00]. When product sales *decline*, a limited servicing of the product with minimal bug fixes replaces the previously more extensive product evolution. To account for the declining income within the decline phase, a product manager seeks ways to reduce cost, e.g. by moving support services to a low-cost country. With the final phase of *withdrawal*, the product manager prepares the product's phase-out and performs its closedown.

The life cycle phase also influences maintenance. A vendor may introduce variability into the product to address diverging needs of different market segments, especially during the growth phase of a product. Later, the product organization will recognize a need to minimize the number of parallel versions of a product that are to be maintained. Excessive parallelism would produce portfolio and organizational complexity. Therefore, the maintenance of older versions is frequently decommitted with some lead time to motivate customers to migrate to newer versions. These migrations are often a point of conflict between vendor and customers that vendors try to mitigate by offering discounted upgrade prices. Challenging decisions are thus required when moving from one phase to another. Approaches that worked well in one phase may need to be changed to prepare for the next phase, even when the changes create irritation and resistance.

In practice, the life cycle of a product is hardly as stepwise and linear as Fig. 4.15 suggests. While the markets for B2B license products may be slow, Internet-based markets tend to be turbulent. Consequently, product development and evolution are often iterative and confronted with much trial-and-error and failures. Also, a product organization may discover that a product is in a different life cycle stage than initially thought. For example, a product believed to be mature may be changed and evolved to address a newly discovered growth market.

The iterative nature of the product life cycle is particularly important until a product is on the market that satisfies customer needs, is well differentiated, and is technologically viable [KhuFrGor15]. Only successful products undergo all life cycle phases; the unsuccessful product should be withdrawn rapidly.

The product organization utilizes measurements that are connected to the position of a product in its life cycle to support product life cycle management. Data from product analysis and market analysis will be used to determine the current product life cycle phase. Other measurements are used to monitor and control the life cycle-specific work and the outcomes that are achieved with that work. Section 3.13.2 offers more details on these measurements.

Life Cycle Model for a Product Category

Products compete within product categories which Moore defines as "A term used by customers to classify what they are buying and distinguish it from other purchasing choices". Examples are smartphones, or ERP software. These product categories either already exist in the market, or a vendor might research or leverage new technologies to establish a new category. The product category also follows a life cycle. For the early stage, Moore coined the term "technology adoption life

Phase	Constellation	Dominant New Customer Type
Early Market	Testing and early adoption of a new product category by a market.	Enthusiasts and visionaries
Chasm		Niche market
Bowling Alley		Increasing number of niche markets
Tornado		Pragmatists
Main Street		
Growth Market	Large-scale adoption of the product category.	
Mature Market	Growth flattens, and competition is noticeable.	Conservatives
Declining Market	Technologies for new product categories emerge.	Skeptics
End of Life	A new product category is in the Tornado stage.	-

Fig. 4.16 Market development life cycle model for a product category based on [Moore08, Moore04a]

cycle", which he defines as "A model that describes how communities react to the introduction of a discontinuous technology, consisting of a progression through five adoption strategies: technology enthusiast, visionary, pragmatist, conservative, and skeptic". The later stages are called "category maturity life cycle" defined as "A model that describes the rise, duration, and decline of a category of product or service". The combination is named "market development life cycle" [Moore08, Moore04a]. Figure 4.16 gives an overview.

The life cycle of a product category reflects how the market reacts to that category. Most customers will be resistant to new types of products and will only adopt what is proven. Technology enthusiasts and visionaries will be more welcoming early on. So in order to make decisions about whom to market a product to and how to market that product at a given point in time, it is important for product managers to understand the current category phase of their product. Spending money in the wrong area would be a waste, and highest potential customer types change from category phase to category phase.

An idea for a new product category is often developed in collaboration with enthusiasts that are willing to try and explore new technologies with low readiness. These innovators are few in numbers. A product needs to address the next class of customers: the visionaries to grow product sales. Visionaries are often interested in a potentially risky product because of the potential it brings to build new business. While working with the visionaries, a product manager must manage to adapt the product to the needs of the first big category of customers: the early majority. If not successful, the product will be trapped in the "Chasm" and cannot get to significant revenue growth.

Customers in the early majority adopt a product on pragmatic grounds. The product should be well established, deliver the expected value, and be of high quality. Moore has characterized a product category at this stage to be first in a Bowling Alley, then a Tornado, and finally the Main Street. Each Bowling pin in

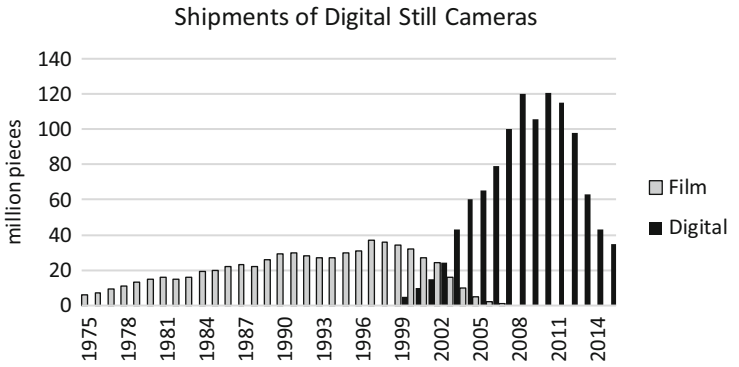

Fig. 4.17 Example of how product categories affect markets: Camera market (Source: CIPA)

the Bowling Alley corresponds to a market niche that may be convinced of the category with the help of another already won niche. The Tornado and Main Street represent the transition to the mass market that offers attractive growth rates with few threats by competitors.

When a product category further matures, the conservative customers become important. They invest in products that are mature and convenient to use. Also, they are price-sensitive. If multiple competitive products exist in the same space, fierce competition may characterize the mature product category because the market growth has flattened.

A market starts to decline when technologies for new product categories emerge and are being tried. In this stage, the only source of new revenue for the aging product category may be the skeptics who have resisted so far. They may be convinced if the product is integrated into other products or services. As soon as a new product category is in the Tornado stage, according to Moore, the declining market should be exited.

Figure 4.17 shows how successions of product categories imply changes in the markets, and how each category evolves measured in units shipped. We chose here the camera market because it is well documented and an interesting example of how the inclusion of software into a product category can replace categories based on other technologies. Growth and decline are clearly evident as one category replaces a previous category. The growth of the digital camera category puts an end to the prospering film cameras category.

Following Moore's model, the growth was achieved with sales to early adopters. These early adopters followed the latest technology that had achieved a good reputation. Once compact digital cameras were sufficiently reliable, the early adopters shifted from buying analog cameras to digital ones. The products based on the previous technology were mostly sold to a declining number of conservatives. The graph also illustrates the decline of the digital still cameras category.

Phase	Value Focus	Value Proposition	Product Planning Priority
Early Market	Curiosity of enthusiasts	New technology to try and explore	Identify most attractive use cases for the product
	New opportunities for visionaries	New means for business development	Mature the product and reputation to win trust of niche market
Chasm, Bowling Alley	Value for pragmatists (narrow niche)	Practical, value-creating product with low risk	Increase convenience and win trust of new niches
Tornado, Main Street, Growth	Value for pragmatists (broad market)		
Mature	Convenience for conservatives	Established, cheap standard with reliable support	Develop products for new product categories
Decline	Necessity for skeptics	Product integrated into other offerings	Phase-out the product, and replace by new product category or exit the market

Fig. 4.18 Implications of the product category life cycle for product planning

The relative maturity of technology for a product category and Moore's market maturity life cycle model affect a product's value focus and proposition as well as the product planning priorities. Figure 4.18 gives an overview. These recommendations apply to any competitor who addresses the concerned market with the concerned product category. The value proposition characterizes the type of message for which new customers will be receptive. The product planning priorities indicate the type of work that should be done by the product organization to sustain business in the market.

4.4.3 Product Planning Tactics

Both, the product and category life cycles influence product planning priorities and focus areas. The development of a new product for a new market will require a planning approach that completely differs from the evolution of an existing product for a mature market. We offer here a discussion of three important constellations. Other constellations are possible and may be addressed with a combination of the three we present here.

Innovating with a New Product in an Immature Product Category

When developing a new product for a new market, the product manager finds himself in the product life cycle stage of product conception and in the early product category cycle stage where a new technology is becoming interesting for enthusiasts and visionaries. This situation is covered by our Powerboat or Ice-breaker product scenarios. The product conception phase implies substantial investment into the development and testing of the new product. Since there is no

established market for the chosen technology yet, much work goes into trials in collaboration with the enthusiasts and visionaries to identify the most attractive use cases for the product.

Many startup companies are working in this constellation. As a result, approaches that facilitate flexible trials and learning influence product management thinking. These approaches may also be applied in an established organization, usually by putting the innovating team in a separate organizational unit. The separation ensures that the team is not concerned with the daily business operations of the rest of the larger organization and that the measurements of the team's performance suit the innovation situation. In both, the startup and the separate unit cases, the product team benefits from business and technological background, either developed at universities or reused from previous products, to accelerate product development and market introduction [KhuFrGor15].

The *Design Thinking* approach describes how to develop attractive product concepts. It suggests how to collaborate with stakeholders by experimenting with prototypes, developing empathy for users and customers, and adopting a systemic view [Brown08]. In a design thinking project, the team identifies a problem that matters, generates ideas for potential solutions, implements and tests these solutions, and brings the successful one to the market.

The *Lean Startup* approach explains how to plan, test, and evolve a business model until one is found that works [Blank13c, Maurya12, Ries11]. The business planning activities center around the Business Model Canvas that we introduced earlier (see Sect. 2.4), which is used to document the building blocks of the product business that the product manager intends to develop. Each of these building blocks represents hypotheses that need to be tested with prototypes or minimum viable products, and adapted to the lessons learned from the tests. The Design Thinking and Lean Startup approaches can combined. An early business model canvas can be built on top of what has been learned during the design thinking work.

The Lean Startup approach builds on *Agile Development* by initiating a work style of rapidly iterating between implementation and testing. In Agile Development, the team uses a backlog to prioritize incremental deliveries of a working product [Schwaber02]. For software-as-a-service products, *DevOps* may be used to enhance the Agile approach. In DevOps, the cross-organizational, multidisciplinary product development team is extended to the organization that hosts and operates the growing product [BasWebZh15, Bosch12].

New Product in a Mature Product Category

When developing a new product in a mature product category, the product manager is in the Powerboat or Icebreaker product scenarios. He finds himself in the product life cycle stage of conception and creation. The new product is to be introduced to a market in which pragmatic customers expect more value and conservative customer prospects expect more convenience. The mature product category implies that competitors exist and have well-established ties with important customers.

New products in a mature product category unavoidably implement a *follower strategy* that copies or imitates a successful product concept. The product addresses

already known needs that are relatively easy to elicit because of the previously existing products. Important for the success of the follower strategy is that the product is more attractive to the targeted customers than the competing products. To achieve this aim, the product manager may adapt a broader product concept to the specific needs of a narrow customer segment or offer a comparable product at a lower price and greater convenience.

Evolution of an Existing Product in a Mature Product Category

When evolving an existing product in a mature product category, the product manager is in the Speedboat or Cruise Ship product scenarios. He finds himself in the product life cycle stage of growth, maturity, or decline, and in the category life cycle stage of the mature market.

The mature product life cycle stage implies that the product is being used as a cash cow to invest in new products. A limited part of the product's revenue is used to extend the product scope and increase the target market. The increasing conservatism of the new customers implies that the feedback obtained from the large customer base should be used to make the product easier to use and to integrate it as a standard into other offerings.

When companies have existing products in mature product categories, product management thinking is usually influenced a lot by methods or approaches that facilitate collaboration with many company-internal stakeholders, the utilization of a stream of feedback about existing products, and structured decision-making with staff that has in-depth product knowledge. The work is often performed by following structured processes that reflect how companies at high levels of maturity do work.

An important process is the continuous product requirements management process used to capture bug reports and feature requests, perform triage, and plan releases according to a funnel metaphor ([Davis05], see Sect. 4.2). With product requirements engineering and release planning (see Sect. 4.3), the product manager aligns the evolving product with the company's strategic objectives [GorsWohl06] and agrees on product development plans with company-internal stakeholders [PhaFarPr07, Carlshamre02].

We have discussed the benefits of simplifying products and services offered to customers. To accomplish this product lines and ecosystems become attractive ways to deal with the increasing diversity of the customer base and new markets. The product organization may utilize *product lines* as a structured approach to reusing components across a set of product variants, embedding them in a common platform, and specifying the selection of features that are offered for each variant ([PohlBoLi05], see definition in Sect. 2.2). The product organization may also allow other companies to use the product as a common platform for the development of own value-adding products and services. This *ecosystem* approach allows sharing risks and efforts with specialized companies against a share of the possible revenue ([JanBrCus13], see Sect. 3.11).

4.4.4 Summary and Conclusions

In this chapter, we have discussed product life cycle management by looking at the product life cycle, the technology adoption life cycle and the category maturity life cycle. We analyzed how these life cycles affect product planning. The product life cycle describes the six phases of a product in a company's portfolio. The first three phases are investment phases. In the following three phases, the product serves as a cash cow to generate profit and finance the investment in new products. With the increasing maturity of a product category, a series of new customer types appear whose specific needs must be addressed if a product in that category is to be successful.

The phases in these life cycles relevant for a particular product affect the product manager's focus and product planning tactics. The development of a new product in a new category differs substantially from the evolution of an existing product in a mature category. We have outlined the role of the Lean Startup approach to address the former constellation and of structuring the stream of customer feedback and requirements to address the latter constellation. The approaches described in this chapter are prototypical approaches. For any real-world situation, our discussion is the starting point for an essential specific analysis which may lead to conclusions that differ from what theories suggest.

4.5 Process Measurement and Improvement

4.5.1 Overview

Many software organizations fight with release schedules, cost control, and release of products that should meet the expectations of the markets. To find ways to improve an organization, assessments are used. With assessments, the organization determines the practices that are in use by the organization and compares them with those of other organizations or frameworks. The results of an assessment may also be compared with benchmarks. A benchmark is quantitative data about the work and the achieved outcomes within the same or another company or even within the industry. The benchmark allows comparison of the assessed organization with an earlier situation or 'best practice.'

This section gives a brief overview on of product planning processes, how product management performance may be measured, what the frameworks are that may be used for improving software product management practice, and what the tools there are to are that facilitate product management productivity. The section adds the process performance perspective to the product performance perspective that was discussed in Sect. 3.13. The section also relates to Sect. 5.6 that explains how the performance indicators may be obtained so that they can be used for managing performance.

Strategy task	Continuous	Periodic	Triggered
Product Life Cycle Management		P for existing products	T if new
Roadmapping		P	T if changes need to be reflected and a presentation is required
Release Planning		P	T if changes need to be reflected
Product Requirements Engineering	C		

Fig. 4.19 Classification of planning tasks

4.5.2 Product Planning Processes

For classification of product planning tasks, we use the categories introduced in Sect. 3.14:

- Continuous (C): done more often than once a month.
- Periodic (P): done monthly or less often, but with predefined frequency.
- Triggered (T): only done when a particular event or request happens, i.e. not with predefined frequency.

We apply this classification to the product planning activities listed in the SPM Framework (Fig. 4.19):

Most software organizations define and implement a process for product requirements engineering in order to stay on top of the usually high number of requirements to be managed. Such a process is typically be supported by an appropriate tool (see Sect. 4.5.4). In mature software product management organizations, processes for release planning and roadmapping are often defined in a more formal way.

4.5.3 Improving Software Product Management Performance

Once a product strategy has been defined, product management is in a race to bring the product to market. Time-to-market is critical to position a product in the market before competitors do. Even if a product has already been released, features must be rapidly developed or evolved. With such product extensions, the product may become attractive for additional segments of the market that were insufficiently addressed before, and more satisfying for existing customers.

Time-to-market also influences the value that is created by the product [DenCle04]. It influences the moment in time when revenue is created with the developed product and also influences the return-on-investment rate. The earlier a product is released the earlier revenue is generated. That revenue can fuel further investment that is needed to build the product. At the same time, more revenue may be generated because of the time advantage against competitors who try to take a share of that revenue. Also, early revenue shortens the time to reach the break-even point as revenue can be accumulated over a longer time than if product management waits with product releases.

When a product is established, efficiency and adherence to plans become increasingly important. The product management organization must be structured and processes designed so that as much value can be created with as little effort as possible. Release plans are a frequent focus for improvement. They must be revised when schedules are not met, whether a product's schedule or schedules on which the product depends, and when the product releases do not perform as expected. Release plans revised outside of the corporate planning cycles are formally synchronized with the next corporate planning cycle.

The product organization should prioritize product development work with maximal value impact and minimal cost [KarlRyan97]. The amount of pending work implied by the product requirements and the time and money spent on these activities can constitute further measurements. Changes in the number of pending product requirements at different steps in their realization (e.g. analysis, decision-making, implementation, testing, and release) can be used to identify bottlenecks or overcapacity in the product organization [PetWoh10].

As soon as a product is released there will be feedback from actual and potential customers. Measurement of confidence in the product planning process may be measured by gauging its ability to predict that feedback before the release of the product. Often, confidence issues may be identified that allow proactive risk management. Common issues concern the linkage of requirements with business strategy, seeing the big picture of the offering, understanding of the planned product's value, and knowledge of customer problems [KKTLD15].

After the release, the customer feedback is essential for deciding whether the product has been appropriately designed. If interest in the product is mediocre, the product strategy, in particular scope, positioning, or target market may have to be changed. Outcome measurements include those related to product development, such as planning accuracy [Ebert07], and those related to how well requirement cost and value were predicted [Karlsson06, HerrDan08]. The better the original assumptions were, usually a consequence of early product validation, the less requirement volatility and the more product stability. A root-cause analysis of the major deviations between prediction and realized results should be used for guiding the future product planning decisions. Examples of causes that lead to deviations are under-estimation of development effort, orders issued by a specific customer, actions from competitors, or the lack of availability of an effective, validated solution to a design problem. Any critique that is received as a feedback should be evaluated and the product definition or positioning adapted.

Performance Aspect	Selected Measurements
Time-to-market	Duration between product or feature definition and release.
Value	Return-on-investment, revenue, market share, customer feedback
Efficiency	velocity in value per time unit, amount of work remaining (backlog)
Predictability	Confidence, adherence to plans
Quality	Requirements volatility, number of problem reports

Fig. 4.20 Aspects of product management performance and corresponding measurements

Figure 4.20 summarizes these aspects of product management performance and offers suggestions for relevant measurements. The may be used as a starting point for planning product management performance measurements and assessing how the performance evolves over time. A root-cause analysis of the major deviations between prediction and realized results should be used for guiding future product planning decisions.

Planned process improvement programs follow the assessment and benchmarks [Jones08]. Various frameworks have been developed for improving product management practice in the product organization. Section 2.5 of this book gives an overview of software product management frameworks that are useful for assessing software product management practice and for planning improvements.

The frameworks are useful for assessing the as-is situation and should be used for planning incremental improvements that are effective but do not overwhelm the product organization. The guidelines that these frameworks contain should be applied consciously and by reflecting what works well and what not in the organization's practice.

Also, it is effective to elicit experiences from product managers on a continuous and to use these for bottom-up improvements of product management practice. Of greatest importance are patterns of recurring practices that work well and are effective. These should be documented and spread throughout the organization.

4.5.4 Tool Support

Product management is such a multifaceted, data-driven, and decision-oriented work that it would not be possible without tools. A product manager documents ideas, analyzes data to answer questions about the product, leads the product organization through important decisions about the product, and orchestrates product development and operations. Documentation tools are used to make ideas available throughout the organization. Modeling tools are used to analyze situations and reduce complexity. Prototyping tools may be used to allow stakeholders to experience possible products. To win stakeholders and to orchestrate their work, communication and tracking tools are needed. Figure 4.21 gives an overview of important tool categories.

Category	Benefits	Examples
Specific Product Management Tools	Support of product planning tasks, in particular roadmapping, some extended into product strategy tasks	Aha!, Product Plan, ProdPad, SensorSix[1].
Documentation Tools	Documentation of ideas and decisions and visualization of concepts. The tools should allow collaborative editing of the documents.	Word processors, presentation tools, e-mail, Wikis like Atlassian Confluence, and shared repositories like Dropbox.
Modeling Tools	Structuring and analysis of data, information, and knowledge in preparation of decision-making. Some of the tools allow informal modeling without any constraints, while others implement standard languages.	Mind-mapping, roadmapping, and system modeling tools, e.g. with UML or SysML. Stackoverflow ranks UML tools[2].
Prototyping Tools	Approximation of systems, their user interfaces, and their use for discussion and testing with stakeholders.	GUI design tools, e.g. for creating wireframes. Quora has a discussed ranking of wireframing tools[3].
Communication Tools	Communication with stakeholders. These tools are usually used in conjunction with documentation tools.	Phone, e-mail, messengers, and conferencing tools, such as Skype or GoToMeeting.
Tracking Tools	Enactment of workflows, usually to manage requirements, issue, and task backlogs.	Spreadsheets and issue management tools. MakingOfSoftware offers a curated overview of such tools[4].

Fig. 4.21 Categories of tools for product management

The market for specific product management tools is quite young. Historically, product managers have used development tools like Jira even though they are not ideally suited for product management work. At least the use of development tools ensured that there were no data synchronization problems between Product Management and Development. The newer specific product management tools try to address that requirement by providing interfaces to popular development tools like Jira. Examples are Aha!, ProductPlan and ProdPad. The requirements management tools category is much more mature. IBM Rational Doors succeeded in establishiing itself as a requirements management tool. It has become a standard in the automotive and aviation industries. For markets that are not required to comply with comparable regulation, a new market leader, Atlassian, has emerged.[8] Atlassian offers cloud-based products for Wiki-based requirements definition and requirements management.

Tools from domains other than software have not been successful in capturing the market for software product management. For example, there are components in Enterprise Resource Planning (ERP) software offerings called product life cycle management (PLM). These were designed primarily for manufacturing companies

[8]Gartner's 2015 Magic Quadrant for Application Development Life Cycle Management.

where product life cycle has a meaning very different from software. We expect new categories of tools to become important in product organizations, including product usage monitoring tools and collaboration tools that allow developers, supporters, and users to interact with each other. Among these new tool categories, there is no dominant tool yet, and we expect that more tools will be released and disappear again.

Since product management works intensively with other units within the company, it can be helpful for a product manager to get access to these units' task-specific software systems. Whether to grant such access will be decided in the context of the company's culture, the organizational structure and defined role of product management, as well as the personal working habits and reputation of the product manager. Project management tools in development, issue tracking tools in development and support, customer relationship management tools in marketing, and sales and planning and controlling tools of the company or individual units can be of interest. What is important is that the product manager has timely access to all factual data that he needs for his work. If this can be achieved without giving him access to the task-specific software systems, everyone will likely prefer that.

The use of tools can significantly improve productivity. At the same time, there is no guarantee that productivity improvements will be achieved. The effect of a tool on productivity was shown to differ significantly depending on the context of where the tool was used [BrMaJaHe96]. A tool may increase the productivity for some products and projects and, at the same time, decrease the productivity for other products and projects of the same organization. Also, the processes run in the organization and the complexity of the products affect the impact of a tool.

Conversely, the choice of an inappropriate tool can have devastating effects on an organization. Employees may be blocked in their work due to an inappropriate tool and will do everything they can do to circumvent the tool [Farmer06]. A systematic tooling process should be followed to avoid such problems proactively [GoWoGL06]. The selection process should select tools which address the right problems, are better suited than other tools, can be implemented in the organization, and are accepted by the users that interact with the tools in their daily work. All tools should be integrated into the organization's toolchain to avoid manual, clerical work and find a permanent home in the organization [Farmer06]. Only tools that fulfill these requirements and lead to the desired productivity improvements should be rolled out organization-wide.

4.5.5 Summary and Conclusions

This chapter has given an overview of indicators for measuring the performance of the product management process and how the performance may be improved. The indicators reflect the important primary concerns of a product manager: time-to-market, value, efficiency, predictability, and quality. Frameworks for process improvement and software tools are instruments to improve the performance of product management. These become effective when the product manager knows

how to integrate them into an appropriate methodology—knowledge that may be obtained by education such as defined by the ISPMA syllabi.

Process improvement frameworks and classroom training need to be accompanied by practical experience gained by the actual execution of practicing product management and learning from others. Observing and reflecting on one's practice, attempting at to generalizing generalize from the observations, and testing the learned lessons in new situations provides relevant advancement of understanding and abilities [Kolb14]. Communities of practice (like ISPMA) allow practitioners to developing relationships with experts, peers, and stakeholders, sharing share ideas, setting standards, and building tools to solve problems [WenSny00]. Such learning enables product managers to improve as individuals, and as a community, and as effective members of their companies.

Strategic Management

5

Strategic Management is an activity within an organization with the objective to define, plan, agree, implement and evaluate the organization's strategy. It is part of the responsibility of executive management who can delegate preparatory work to staff functions. Strategic Management includes a number of elements related to software product management (see the ISPMA SPM Framework in Sect. 2.5). Software product managers are typically not responsible for any of these activities, but they either participate in them, e.g. portfolio management, provide inputs, or make use of their outputs, e.g. product analysis.

Of course, it is not the objective of this book to provide a handbook on executive management. This is covered by a huge spectrum of publications. However, since a software product manager has the responsibility for his product(s) and thereby a partial responsibility for the success of the whole company, he is very directly involved in some aspects of executive management.

A Software Product Manager usually spends some of his time with the task to represent his product in the internal strategy and planning processes of his company. This includes the marketing and sales plans and the budget and resource planning. The underlying question is which resources will be dedicated to the product in the short, medium, and long term. This decision is based on market and revenue forecasts, the positioning of the product in its life cycle, and the dependencies with other products. From these elements, the product manager puts a "story" together that is used to "sell" the product within the planning processes.

The company's culture influences how these planning processes work and what is expected from the product manager. Ideally, all involved parties should have the common goal to get to an agreed result that is good for the company. Often, however, these processes degenerate into a competition that the players try to use for their personal advancement. The "winner" is the one who gets most of the resources for his product. Only the executive management can prevent this degeneration. The individual product manager will have to play his role according to the company's culture, for the good or for the bad.

H.-B. Kittlaus, S.A. Fricker, *Software Product Management*,
DOI 10.1007/978-3-642-55140-6_5

Typically, the corporate strategy and planning process is a mix of bottom-up and top-down planning. Bottom-up means that each product manager develops a plan from his product perspective. Top-down means that executive management, typically under the lead of finance, looks at the aggregated bottom-up plans and cuts them down to what seems affordable. Since the assigned budget and resources have their consequences on the revenue side, this process is iterated until an agreement is reached. This process serves as a synchronization point at which the plans on all levels and of all products are synchronized. Executive management usually defines the schedule of this process. IBM, for example, goes through two cycles per year, the Spring and the Fall plans.

At this point we want to go into the elements of the corporate strategy and planning process and the role that the product manager plays in them.

5.1 Corporate Strategy

5.1.1 Overview

Corporate strategy is a phenomenon that entered the scene in the sixties of the twentieth century. Since then, many different approaches and tools for strategic management have been developed and are used across industries. These approaches and tools can be traced back to different schools of thought that evolved over time. Some of these tools and approaches are frequently used in modern software organizations. We will focus our discussion on these frequently used tools.

In addition to industry-agnostic approaches to corporate strategy, there are also tools and approaches that have been developed specifically for high-tech markets, including software markets. They have been designed to address the specific strategic challenges of markets that are based on quickly evolving technology, resulting in fast value erosion for products.

Software product managers may have to provide input whenever the corporate strategy is updated or revised, and they need to ensure that product strategies stay consistent with corporate strategy. To achieve this, software product managers need to be aware of the strategy tools and approaches that are used in higher-level strategy processes in their organization. Understanding which key ideas and assumptions are underlying the use of these tools and approaches helps product managers improve their contribution: they can provide more useful input into strategy processes performed at higher levels in their organization and will better understand and use the guidance they receive from these higher-level strategy processes.

5.1.2 Concept

Corporate strategy considers a timeframe that is at least as long as the strategic timeframes of the individual software products. Therefore, the timeframe considered may be up to 5 years, or even longer, depending on the domains covered.

Strategy processes on the corporate level can be triggered or happen on a periodic schedule. A strategy revision may be triggered by major changes in the environment, for example substantial regulatory changes, unexpected major strategic moves by competitors, or technology disruptions. Updates to an existing strategy are typically conducted periodically—tied to the organization's regular planning and reporting cycle, for example preceding the annual planning cycle.

A corporate strategy process includes activities to develop or update the following strategy elements: corporate vision, mission, values and goals, corporate positioning, business model and financial plan, product portfolio and its evolution, resource and competency evolution, technology trends and innovation strategy, market trends and competitive strategy, policies and governance.

Many of these strategy elements are comparable to corresponding elements on the product strategy level. Figure 5.1 compares the elements from these two different levels of strategy process.

However, there are a few differences beyond the scope of product(s) or businesses being covered. The corporate vision is typically a very short statement, often just one very high-level sentence about a desired future state that the corporation wants to help create. This describes why the company exists. To further substantiate this, it typically is accompanied by a corporate mission that describes on a high level what the company is doing to achieve the vision, plus a statement of the company's values and goals.

Fig. 5.1 Elements of corporate strategy with their equivalents on the product level

Elements of Corporate Strategy	Comparable Elements on the Product Strategy Level
Corporate vision, mission, values and goals	Product vision
Corporate positioning	Product positioning
Business model(s) and financial plan	Business model(s) and financial plan
Product portfolio(s) and their evolution	Product Roadmap
Market trends and competitive strategy	Market trends and competitive strategy
Technology trends and innovation strategy	Technology trends (from market analysis)
Resource and competency evolution	–
Policies and governance	–

Instead of a product roadmap, the corporate strategy process will look at entire portfolios of products and set goals and boundary conditions for the evolution of these portfolios.

Finally, corporate strategy addresses three areas that do not have a direct equivalent on the product strategy level: Innovation strategy (see Sect. 5.3), resource and competency evolution (see Sect. 5.4), and policies and governance.

5.1.3 Process

Corporate strategy processes are often based on industry-agnostic tools and approaches which can be traced back to different schools of thought that have evolved over time.

In [MinAhlLa08] Mintzberg, Ahlstrand, and Lampel provide an overview on strategic management approaches, identifying ten underlying schools of thought in corporate strategy. They further classify those ten schools into three major groups: prescriptive, descriptive, or integrative schools of thought.

Figure 5.2 provides an overview of all ten schools of thought and their associated tools, based on [MinAhlLa08].

The tools and approaches highlighted in the last column of Fig. 5.2 are frequently used in strategic management processes:

- **SWOT Matrix**: maps internal **S**trengths and **W**eaknesses of the organization against the **O**pportunities and **T**hreats presented by the external environment. This is frequently developed for individual products as well, as part of the market analysis (see Sect. 5.5)
- **Scenario Planning**: aims to broaden the view of decision makers by developing several alternative long-term scenarios. Each scenario describes a possible future state of the organization's external environment. For each scenario, the impact on the organization and possible responses are elaborated. This is especially suitable for volatile times and fast-moving markets—that's why this strategic tool is sometimes used by software organizations. Due to their market understanding, product managers may be asked to contribute to the development of scenarios, or they may contribute to developing the strategic responses in the various scenarios.
- **Porter's 5 Forces**: Michael Porter in [Porter79] and [Porter08] identifies five forces that characterize the nature of competition within an industry, also called the industry structure. The five forces are: threat of new entrants, bargaining power of customers, threat of substitute products or services, bargaining power of suppliers, and rivalry among existing competitors. In [Porter85] he described that only three classes of strategies can be chosen by firms—what he calls *generic strategies*. These are cost leadership strategy, differentiation strategy, and focus strategy (niche strategy). The 5 Forces model and the generic strategies are routinely taught in management education and are broadly known and used, especially in industries where the costs for manufacturing

School	Key assumptions	Key authors	Key approaches & tools
Prescriptive – how strategy *should* be formulated: strategy precedes structure			
Design School strategy formation as a process of *conception*	"Establish fit" – between internal capabilities and external possibilities; Design several alternative strategies (a creative act) and choose the best	Kenneth Andrews	**SWOT matrix** (internal **S**trengths & **W**eaknesses, external **O**pportunities & **T**hreats)**;**
Planning School strategy formation as a *formal* process	"formal procedure, formal training, formal analysis, lots of numbers" replace the creative act of strategy design	H. Igor Ansoff, George Steiner	Elaborate planning cycles and schedules, **cascading systems of plans**; **Scenario planning**
Positioning School strategy formation as an *analytical* process	Impact of industry structure on strategy: only a few positions in the market are desirable, and there are only a few generic strategies to select from	Michael Porter	**Porter's 5 forces** - for competitive analysis **and Generic Strategies**: cost leadership, differentiation, focused strategies; Experience curve => focus on market leadership **BCG growth/share matrix** – for portfolio management; Value chain analysis
Descriptive – understand strategy *as it unfolds*			
Entrepreneurial School strategy formation as a *visionary* process	Strategy exists in the mind of the leader (entrepreneur) as a vision, strategy formation is rooted in experience and intuition of the leader; often starts in niche market	Schumpeter (creative destruction)	Vision statements
Cognitive School strategy formation as a *mental* process	Strategy formation as a cognitive process that takes place in the mind, creating perspectives that shape how people deal with input from the environment	Many different sub-schools and authors	Tools to help managers/ leaders better understand their cognitive biases and their personal preferences, e.g. by doing a Meyers-Briggs personality test
Learning School strategy formation as an *emergent* process	Strategies emerge as people (individually or collectively) learn about a situation as well as their organization's capability to deal with it.	Brian Quinn, C.K. Prahalad, Gary Hamel, Peter Senge,	**Internal corporate venturing;** Learning organization

Fig. 5.2 Schools of thought in strategic management—the big 10

	The leader's responsibility is not to preconceive deliberate strategies, but to manage the process of strategic learning.	& many others	
Power School strategy formation as a process of *negotiation*	Strategy formation is shaped by power and politics, both inside the organization and outside. The resulting strategies take the form of positions or ploys more than perspectives	Many, including Michael Porter	**Strategic alliances** **Strategic sourcing** - incl. make vs. buy and vertical (dis-) integration decisions; Stakeholder analysis; Strategic maneuvering - in response to competitors
Cultural School strategy formation as a *collective* process	Strategy formation is a process of social interaction, based on the beliefs and understandings shared by members of an organization	Several sub-schools	Strategic resources
Environmental School strategy formation as a *reactive* process	Leadership is a passive element for reading the environment and ensuring proper adaptation by the organization	Various authors and sub-schools	--
Integrative			
Configuration School strategy formation as a process of *transformation*	Organizations are stable most of the time, but occasionally, they need to transform – take a quantum leap to reach another configuration. Strategic management needs to sustain stability most of the time, but recognize the occasional need for transformation and manage it without destroying the organization	Many, including Mintzberg	Change management

Fig. 5.2 (continued)

and delivering products are non-negligible, for example for software-intensive products that include hardware.

- **The BCG Growth/ShareMatrix**: was introduced by the Boston Consulting Group in 1970 to classify products according to their success in the market vs. attractiveness of the market they participate in. Since it is frequently used in portfolio management as well, see Sect. 5.2 for a more detailed discussion
- **Internal Corporate Venturing**: where larger organizations encourage employees deeper down in the corporate hierarchy to come up with new product initiatives. These initiatives then compete for corporate funding, similar to

startups working to raise venture capital. This is usually part of an organization's innovation management strategy (see also Sect. 5.3)
- **Strategic Alliances** and **Strategic Sourcing**: in today's markets companies typically act within a complex web of relationships, for example with suppliers, channel partners, and other partners. In that situation, strategy needs to be developed collaboratively with partners. For a deeper discussion of partner relationships, see Sects. 3.11, and 3.8.

5.1.4 Examples and Variations

So far, we have looked at industry-agnostic approaches to strategic management. In addition to those, software organizations may use approaches and tools that have been developed specifically to address challenges of fast-moving high-tech markets:

- **Category Maturity Model for high-tech markets**: This model described by Moore in [Moore08] helps determine strategic focus areas depending on the maturity stages of the product categories relevant to the organization (see Sects. 4.4 and Sect. 5.3).
- **Strong focus on innovation management**: Software markets are often moving fast, resulting in fast value erosion—this usually leads to a strong emphasis on innovation management (see Sect. 5.3) and on making sure that the product portfolio stays fresh (see Sect. 5.2).
- **Ecosystem strategy**: Software organizations often need to maintain a complex web of relationships to other players in the ecosystem(s) they participate in. In that case, determining the role the organization wishes to play in the ecosystem—keystone, dominator, or niche player—is typically part of Corporate Strategy (for more on these roles and the associated strategies, see Sect. 3.11).
- **Big data and analytics**: in many cases, in particular with SaaS software, software organizations can obtain detailed information on usage patterns and user behavior that helps making strategic decisions.

5.1.5 Outcome and Impacts

The outcome of corporate strategy processes is typically a comprehensive documentation that describes conclusions and next steps, as well as the rationale behind that, i.e. the process used and more detailed information that led to the conclusions. These will be presented in some level of detail to key decision makers of the organization. A simplified subset of the results, focusing on key messages and required changes will typically be circulated further down into the organization.

Software product managers need to understand their organization's corporate strategy as well as the portfolio strategy so they can ensure their product strategy is aligned with these higher-level strategies.

5.1.6 Summary and Conclusions

Corporate strategy processes consider a time frame of up to 5 years, or even longer, depending on the domains covered. For this timeframe, a wide range of strategy elements are developed or updated: from very high level elements such as corporate vision, mission, values and goals, down to policies and governance. Strategy processes on the corporate level can be triggered by major external events, or happen on a periodic schedule, for example preceding the annual planning cycle.

Software product managers may have to provide input whenever the corporate strategy is updated or revised, and they need to ensure that product strategies stay consistent with corporate strategy.

To achieve this, software product managers need to be aware of the strategy tools and approaches that are used in their organization. Tools and approaches that are frequently used across industries include Porter's 5 forces, SWOT matrix, BCG growth/share matrix, strategic alliances and strategic sourcing, internal corporate venturing, and scenario planning. In addition, software organizations frequently use tools and approaches developed specifically for fast-moving high-tech markets: Moore's category maturity model, a strong focus on innovation management, ecosystem strategy, and leveraging big data and analytics to support strategy decisions.

5.2 Portfolio Management

5.2.1 Overview

In any product organization, leaders need to ask themselves regularly: Do we have the right products for future business success? Portfolio management addresses this key question, looking both at the existing product portfolio, and at plans for product evolution and new product development. Software product managers are typically asked to represent their product(s) in the update cycle for the product portfolio.

Portfolio management is a term that is well known in the financial services industry. An investor or fund manager invests the available capital in a diversified way, i.e. in different stocks, securities, real estate etc. The total collection of these investments is called a portfolio. Portfolio management is the management of these investments over time following profit and risk criteria. This same approach can be applied to a set of opportunities to invest in existing and new software products, to decide which products and product development initiatives will receive how much investment over the strategic timeframe.

Since the portfolio management process is concerned with investment allocation, it is typically tied to the regular planning and budgeting cycle of the software organization (periodic activity). In practice, it is often performed on an annual basis.

In the software business, it is especially important to critically evaluate the portfolio on a regular basis, for the following reasons:

- Market needs can change rather quickly: software markets are fast-moving, definitions of market segments can shift over time with new market segments forming, and well-established market segments may become less attractive
- There's always a danger that new competitors may enter: software markets typically have low barriers to entry and the boundaries between market segments are often fluid, so that vendors in adjacent markets can enter "our" markets, for example by extending one of their products.
- Software is malleable, so even existing products can evolve in many different directions—which can easily lead to uncontrolled growth and lack of alignment between portfolio products, which may create portfolio gaps or unintended overlaps between products that result in positioning and sales problems within the portfolio.
- Software organizations need to make sure they have a balanced portfolio with products in different life cycle stages—in particular, they need to ensure that there is always sufficient investment in new products.

The last bullet is related to the unique economics of software products. Software is characterized by relatively low variable costs (cost of goods sold) and high fixed costs. It typically takes several years for new software products to become operationally profitable, i.e. before product revenue covers ongoing product-related expenses, unless customers are willing to pay for part of the development cost (see also Chap. 2).

Therefore, software organizations need to make sure they invest some of the revenue surplus of more mature, successful products into new product initiatives. They need to make these investments early enough—so that the new products become mature as revenue from the older products stagnates or declines. On the other hand, a successful mature software product is usually highly profitable: it will have high profit margins (profit as % of revenue) and due to its high overall revenue will be an indispensable source of profits for the organization. Therefore, the organization needs to allocate sufficient investment to mature products to keep them competitive, so that the associated revenue and profit opportunity can be exploited as long as possible.

5.2.2 Concept

The portfolio management process reviews the product portfolio whether it still meets corporate objectives and guidelines, covering both existing products and proposed new product initiatives.

The review evaluates the product portfolio from several angles, asking the following questions:

- Do portfolio products meet their respective measures of business success, such as profitability, market share, number of active users?

- Is the portfolio innovative enough: do we "keep up" the existing products and do we have enough new products in the pipeline?
- How do we want to evolve the portfolio: are there new opportunities that we'd like to take advantage of—with new products or major extensions of existing products? Do we have gaps in our portfolio, for example due to a new market segment that is emerging?

As input into that process, software product managers typically are required to provide the following product-specific information:

- Product roadmaps
- Forecasts of relevant business metrics, for example a multi-year revenue forecast
- And the investment requested for the product.

In addition, product managers are typically asked to provide a summary of the product-specific market analysis, covering market sizing, trends, and competition (see Sect. 5.5), as well as a summary of the product analysis, describing where their product stands against plan (see Sect. 5.6). These inputs may be used by the portfolio management team to complement and extend the market and product analysis they create on the portfolio level.

5.2.3 Process

Portfolio Management for software products follows the same basic methods and processes as any portfolio management. Based on a structured and transparent process, it balances limited resources in order to maximize benefits.

We emphasized already that due to the fast-moving nature of software markets, innovation is a key concern in software portfolio management. To ensure a focus on innovation, software portfolio management often uses the concept of three time horizons that Moore applied specifically to fast-moving high-tech and software markets [Moore14].

According to Moore, "Horizon 1 corresponds to managing the current fiscal-reporting period, with all its short-term concerns, Horizon 2 to onboarding the next generation of high-growth opportunities in the pipeline, and Horizon 3 to incubating the germs of new businesses that will sustain the franchise far into the future."

To ensure long-term success of the organization, the portfolio should be balanced in the sense of having all three time horizons adequately covered.

Moore contends that Horizon 2 initiatives are the most challenging ones: Horizon 1 covers the existing products which are usually sufficiently equipped resource-wise across the organization, from development to sales. Horizon 1 investments typically deliver a return on investment in the same reporting period they are incurred in. Horizon 3 initiatives are usually conducted in separate lab or research organizations that have their own, separate funding.

In contrast, Horizon 2 initiatives are somewhat lost in the middle: they still need nurturing and investment to become successful products or new businesses, and this nurturing is not part of the research lab's agenda, it needs to come out of the established businesses. However, the full benefits from these investments will be reaped in the future, typically several years out, and not in the current reporting period.

Therefore, organizations frequently fail to adequately create and nurture Horizon 2 initiatives. To address this problem, portfolio management processes typically look at investment proposals for each timeframe separately, and with a special focus on Horizon 2 initiatives. It can be helpful to allocate budgets to the three horizons top-down upfront, and then do portfolio management for each separately.

To achieve an adequate balance between timeframes, organizations may start with a top-down split of the total investment budget between these timeframes, see also Sect. 5.4 on Resource Management.

5.2.4 Examples and Variations

A key challenge in portfolio management is the need to look at many different products at once, to put them in context, and finally, to prioritize them. To help deal with this complexity, matrixes are commonly used to classify the products and get an overview on the portfolio. These matrixes are typically based on two attributes that are relevant to decision making in the portfolio process.

A popular example is the Growth Share matrix. It was introduced for corporate portfolio management by the Boston Consulting Group in 1970 [Hender70] and is still widely used today. It classifies products according to their success in the market vs. attractiveness of the market they participate in. As indicator for market success, relative market share is used, i.e. market share relative to the number three player in the market. As indicator for market attractiveness, market growth is used (High/Low).

The result is a two by two matrix. According to Henderson [Hender70], the four quadrants of the matrix carry the following meaning:

- "Products with high market share and slow growth are cash cows. Characteristically, they generate large amounts of cash, in excess of the reinvestment required to maintain share.
- Products with low market share and slow growth are pets. They may show an accounting profit, but the profit must be reinvested to maintain share, leaving no cash throwoff. The product is essentially worthless, . . ."

  Today, these are often called dogs.
- Low market share, high growth products are the question marks. They almost always require far more cash than they can generate.

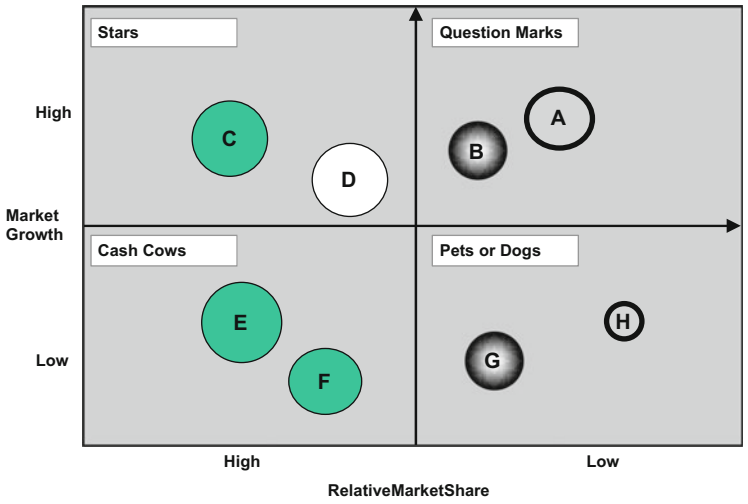

Fig. 5.3 Growth Share Matrix for existing product portfolio

- The high share, high growth product is the star. "... If it stays a leader, ... it will become a large cash generator ... The star eventually becomes the cash cow, providing high volume, high margin, high stability, security, and cash throwoff for reinvestment elsewhere."

The resulting matrix may look like shown in Fig. 5.3. In this example, the matrix represents the entire software portfolio, the color indicates different product families, and the size of the circles represents investment planned for the current year.

In fast-growing and fast-changing markets, portfolio management needs to ensure the portfolio always has cash cows and stars, it needs to critically evaluate the potential of the question marks, and to aggressively exit the pets or dogs.

Many different attributes can be used to build portfolio matrices. In [Cooper00], the following attribute pairs are suggested:

- Risk vs. reward
- Technical vs. market newness
- Technical feasibility vs. market attractiveness
- Competitive position vs. attractiveness
- Cost vs. reward
- Cost vs. time to implement.

Another example that may be used to compare new product initiatives only (Horizon 2) is the Oyster/Pearls matrix that classifies new initiatives by their probability of success vs. expected profit. It may look like this (Fig. 5.4):

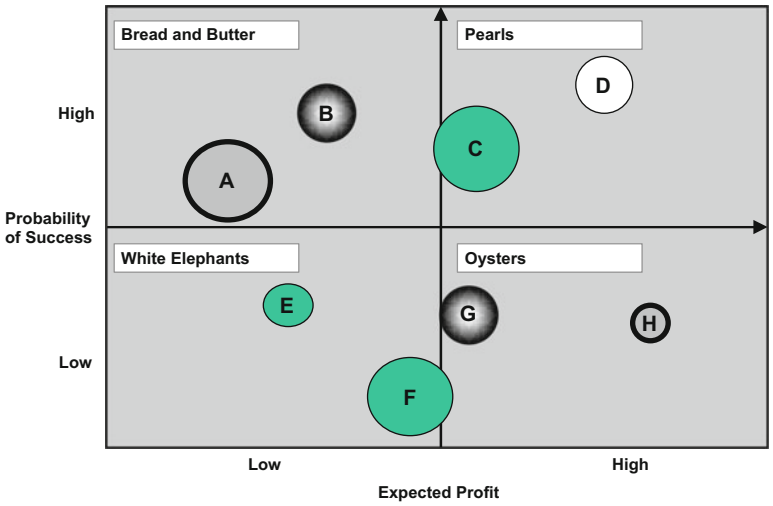

Fig. 5.4 Oyster-Pearls Matrix for new product initiatives

Based on this representation of new product initiatives, the portfolio management process needs to make sure investment in the white elephants is curbed: these are new product initiatives that are not likely to succeed, and even if they were to succeed, they are not likely to generate significant profits.

In project-oriented organizations, portfolio management is often applied to the project portfolio. For product organizations, we do not recommend to do that. When you have applied portfolio management to the product portfolio, there is no need to do portfolio management for the projects in which new releases and versions of these products are developed.

5.2.5 Outcome and Impacts

On a portfolio level, one of the desired outcomes is an investment strategy that minimizes risks on the portfolio level and balances the need for short-term profit maximization with the requirement to invest for future success. Another desired outcome is alignment between portfolio products so that synergies can be exploited, for example by optimizing products for upsell and cross-sell opportunities between adjacent products.

Overlaps between portfolio products—in terms of multiple products offering similar value propositions to the same customer groups—need to be managed carefully. Again, it's a matter of careful balancing: while it may make sense for adjacent products to have some overlaps in their value propositions, so that each product is complete and can be successful on its own, portfolio management usually tries to avoid full, direct competition between products within the same portfolio.

The results of the portfolio update cycle define boundary conditions for the products: In addition to the investment level that will be allocated to the product, product managers will also receive key business goals to achieve, along with target numbers for key business measures, for example revenue or growth targets. Finally, the portfolio management process may define portfolio themes that must be factored into release plans and roadmaps for the product. These boundary conditions, as well as the investment level allocated to the product can have significant consequences for the individual product strategy.

5.2.6 Summary and Conclusions

Portfolio management uses a structured and transparent process to balance limited resources to maximize benefits across existing products and initiatives for new product development.

A key outcome of portfolio management is investment allocation among competing initiatives in the product portfolio. Therefore, portfolio management is typically tied to the regular planning and budgeting cycle (periodic activity, often performed annually).

In the fast-moving software business, it is especially important to critically evaluate the portfolio on a regular basis to ensure the portfolio still is aligned with market developments, meets organizational goals and objectives, and is balanced across the life cycle stages of products.

Software product managers are typically asked to represent their product(s) by providing inputs such as: product roadmaps, forecasts of relevant business metrics, for example a multi-year revenue forecast, and the requested investment. In addition, product managers typically need to provide a summary of the product-specific market and product analysis.

Portfolio management frequently uses matrixes to classify products and to derive appropriate strategies for each class of products. A popular example is the BCG Growth/Share matrix.

One of the outcomes of portfolio management is an investment strategy that minimizes risks on the portfolio level and balances the need for short-term profit maximization with the requirement to invest for future success. Another desired outcome is alignment between portfolio products so that synergies can be exploited, while overlaps between portfolio products are managed carefully.

5.3 Innovation Management

5.3.1 Overview

Software markets tend to be fast-moving with low barriers to entry. Therefore, they are often highly competitive. Differentiators of software products tend to have a short life time, as competitors are quick to catch up.

This results in fast value erosion for software products: delighter features (from the Kano model, see Sect. 4.2) are quickly taken for granted, turning into performance or even must-have features. Competitive advantage of a software product needs to be constantly re-created—and innovation is one way to address that challenge.

That's why software organizations typically put a strong focus on innovation, and why software product managers need to understand key innovation concepts: so they can ensure their product benefits from innovation initiatives of their organization.

5.3.2 Concept

Innovation can occur in many shapes or forms: There can be innovation in how to market products, how to expand current business models, or how to improve organizations or processes. Product managers often focus on product innovations that result in new features, new quality aspects, or an improved user experience.

Innovations also have a different level of market impact, ranging from incremental improvements of the current product offering up to disruptive innovations that create new product categories and new markets, replacing incumbent products in the process.

With such a wide spectrum of innovation types to consider, it is important for software product managers to focus their energy and the available investments on those innovations that are most effective. In [GoFrPaKu10], an innovation process is described that works in software environments.

The suitability of an innovation depends not only of the lifecycle stage of the product itself, but also on the maturity stage of the product category. Geoffrey Moore in [Moore08] (see also Sect. 4.4) identifies fourteen different types of innovation relevant to product managers of high-tech products—including software. He presents a model which innovation types to use at a given category maturity stage. For example, application innovation—finding and exploiting a new application or use for an existing technology—is necessary for a new technology product category to achieve initial penetration into the mainstream market. Line extension innovation—creating a new sub-category to engage new customers or to re-engage old ones—helps maintain and even grow revenue in a mature product category.

Since software organizations put a strong focus on innovation, we may find innovation initiatives at different levels of the organization. For example, a large organization might fund a corporate research initiative with the charter to work on "horizon 3" innovations (see Sect. 5.2), which require several years to turn into viable products. It might also fund "horizon 2" innovations that can be productized faster, but still require more than 1 year to pan out. These might be funded through the portfolio management process. Finally, on the individual product level, "horizon 1" innovations can be funded that require <1 year to be productized.

Even with adequate funding of innovation initiatives for the different time horizons, software organizations often find it a challenge to actually benefit from these initiatives. A famous historic example is the failure of XEROX to benefit from the groundbreaking innovations of its horizon 3 research lab XEROX PARC: these innovations included the windows-based graphical user interface (GUI) and desktop paradigm and the computer mouse.

Therefore, it is critical to align innovation management with the corresponding elements of the corporate strategy on a continuous basis. Alignment is required in both directions: When innovation management leads to significant results these need to be incorporated into the relevant corporate, portfolio, and product strategies to transform them into competitive advantage. On the other hand, changes of the corporate strategy need to be reflected in innovation management to align agendas and resources with the new direction.

5.3.3 Process

Creating innovation is a process of understanding problems, available technologies and creating the right ideas to bring them together. It is extremely difficult to order or formalize the innovation processes.

However, an environment can be created to foster innovation. An important element of that is the organization's culture: an innovation-friendly culture needs to be established and popularized from the top down, where employees are encouraged to come up with ideas, where it is acceptable for innovative attempts to fail without punishing or critiquing the employees, but rather learn from the failure. Processes must be established that allow for the testing and tweaking of innovation ideas.

While it is important to create an environment that fosters innovation, it is also important to have some gates within the company to select the most promising ideas and put some focus on them or reject ideas for which the company may not have the right competencies to implement and is not willing to invest in building up the missing competencies. Overall, this is a challenging task within a company. On one hand, it is necessary to generate many ideas and give them a chance; on the other hand it is important to focus on a few to bring them to success. After all an idea only generates value to a company if it is implemented or sold to someone else.

A great idea is normally never perfect at the beginning and requires many improvement and testing iterations before it matures. This refinement and testing process involves iterations of discussions with customers and the R&D team, creating incremental improvements that are best applied in prototypes, re-testing the concepts and challenging the value. This process can be supported by combining agile development process with customer collaboration.

While creating the innovative environment as well as the decisions which ideas to realize is the responsibility of management, software product managers have the obligation to take advantage of such an environment, spend time for their research in terms of opportunities, come up with ideas together with the team and prepare

them well to increase the chances of receiving funding and support for a project. This includes drawing the big picture about the result once the idea has been implemented. Furthermore, it also requires preparing an initial business case for a first overview of the potential financial impact. Other benefits may need to be mentioned and accounted for as well such as user experience, customer satisfaction or customer retention. A good selection methodology is the software value map, as it incorporates many different value aspects for a structured decision.

Gathering this information supports management in making decisions, validating also that a proposed initiative fits into the overall corporate strategy. Once a project is approved or proceeds to the next gate, it is the product management's responsibility to select the right users/customers to collaborate with. This is necessary to drive development teams to iteratively work on the project and get quick feedback from real users. These many validation steps ensure quick learnings and corrections before significant investments are made and costs are encountered. It is also important that at every stage gate the progress is being presented and the predictions are being updated. The further a project is, the more reliable the predictions of business impact will become. This is to ensure that should something go wrong or deviate from the corporate strategy, a decision can be made to either stop the project, align the project with the corporate strategy or even to expedite the project.

5.3.4 Examples and Variations

Lean Startup

We already emphasized the importance of refining ideas through an iterative process that relies on fast feedback loops with customers. This is especially important for innovations that seek to establish a new product category. An approach to address this special situation has been developed in the Silicon Valley—and the term Lean Startup was coined in 2011 to describe how this approach (see [Ries11, BlanDorf12, Blank13c]). Despite its name, this approach applies not only to startups, but to corporate innovation projects as well.

Driving the iterations is clearly the role of product managers. They act as facilitators to bring the customer demands together with the developer's solution and refine them until the value and user experience are optimal. Requirements triage (see Sect. 4.2) is a simple yet efficient tool for understanding which of the suggested ideas are the most suitable and promising. In this phase, it may also turn out that the chosen strategy to achieve the vision is not appropriate. This leads to pivoting, which means the vision remains but a completely different strategy to get there is required.

Idea Generation

There are many methods that support idea generation. Among them, Cooper and Edgett in [CooEdg09] have done research on the effectiveness of different idea generation methods.

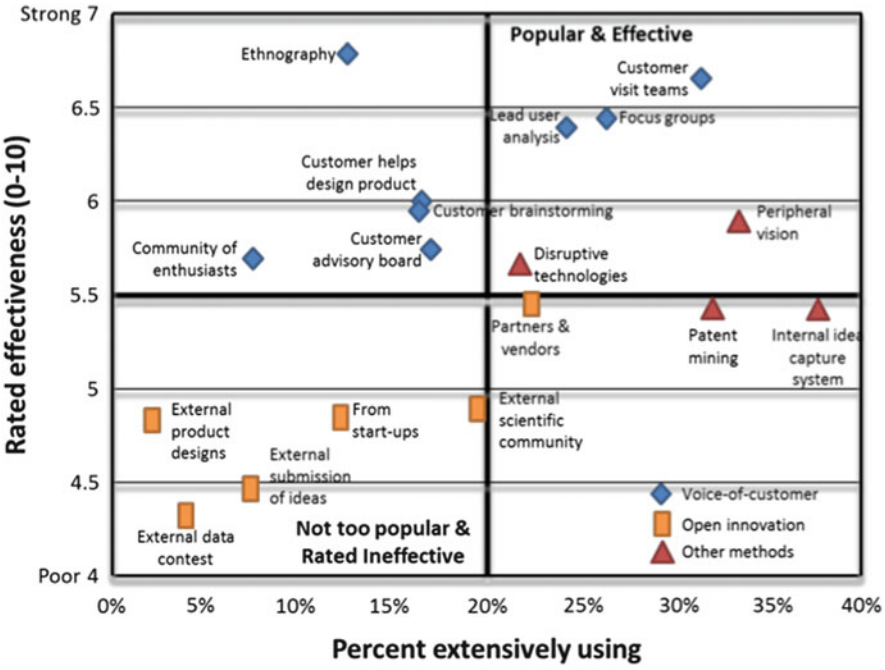

Fig. 5.5 The effectiveness vs. popularity of ideation techniques [CooEdg09]. Used with permission

Figure 5.5 visualizes their findings, mapping idea generation methods by effectivenss vs. frequency of use. The method that is rated most effective is ethnography, which basically means in this context that a product manager or a group of people visit users of a product or performing a job and observe, study and systematically record without interfering the behavior of users, their challenges and possible improvements in their tasks.

In general, the most effective methods involve customers for example through interviews. On the other hand, the most practiced method is internal idea creation. This is the easiest to execute as someone only has to think of an idea without putting in an effort to visit customers. However, as the result also shows, it is not the most effective idea generation method.

5.3.5 Outcome and Impacts

We already discussed that innovation initiatives can typically be classified based on the time horizon they look at. Horizon 3 initiatives are often driven in some type of corporate research lab, while horizon 2 initiatives may be executed within a business unit and funded through the portfolio process. Horizon 1 initiatives are often driven on the individual product level—although they may affect multiple

products and may get "imposed" on the individual product as part of portfolio themes (see Sect. 5.2).

As a result, innovation management impacts the organization on all levels—and it is most closely related to strategy at multiple levels of the organization: corporate strategy, portfolio strategy, and product strategy.

On the product level, the responsibility for aligning with innovation efforts falls on software product managers—they need to ensure that their products benefit from innovation initiatives. Again, this is a bi-directional process: on one hand, software product managers need to understand their organization's innovation initiatives to determine whether results can be used to benefit their products. On the other hand, they may seek to influence the agenda of innovation initiatives so that they address actual use cases and customer problems: Corporate innovation initiatives are often quite interested in leveraging the deep market insight and customer understanding of product managers to help inform their agenda.

5.3.6 Summary and Conclusions

Since software markets are fast-moving, software organizations typically put a strong focus on innovation so their products and product portfolios stay competitive.

Larger software organizations often establish different types of innovation initiatives that work on different time horizons: from corporate research initiatives with the charter to work on "horizon 3" innovations, to "horizon 2" innovations that can be productized faster, and "horizon 1" innovations which typically are driven on the individual product level.

Software product managers are responsible to ensure that their product benefits from the innovation initiatives in their organization. To do that effectively, they need to understand key innovation concepts, such as

- The 3 horizons framework.
- The category maturity model that suggests which types of innovations are most critical depending on the maturity stage of the market.
- Approaches for iteratively improving innovations through iterative processes that relies on fast feedback loops with customers, combining agile development process with customer collaboration and using Lean Startup techniques.
- Idea generation methods, in particular voice-of-customer methods, such as ethnography.

5.4 Resource Management

On the corporate level, resource management needs to ensure that resources are available in the required quantities and qualities and at the required points in time so that the company is enabled to implement the corporate strategy and the aligned

product strategies. This applies to human resources, physical resources as well as information resources. For software, human resources are the most important ones, both in terms of numbers and skills. A software product manager, usually in close cooperation with the responsible line managers, needs to ensure that the resource requirements that result from the product strategy and plan can be fulfilled, i.e. are aligned with corporate resource management.

A product manager's life would be easier if he could make any sourcing decisions by himself based on the product strategy and the annual budget allocated to the product (see Sect. 3.8). However, that is not the way it works in most companies. Decisions on hiring new employees or making investments in IT equipment or real estate or renting space in different locations are considered as long-term commitments that cannot be made solely based on short-term resource needs. So companies usually establish corporate decision processes for these resource aspects in which the individual product manager is a requestor, but not the decision maker. When there are corporate guidelines for sourcing, a product manager may be a bit more empowered within those guidelines. When external human resources are needed for capacity or skill reasons (see Sect. 3.8), additional corporate rules may apply, e.g. procurement processes that are optimized to keep external spending as low as possible. The efficiency of all these processes can differ significantly from company to company. In other words, it can eat up a lot of a product manager's time and energy to "fight the system".

If portfolio management does not only allocate budgets, but also assigns human resources to product teams, that can help the efficiency. For strategic and/or successful products, the core product team should stay quite stable over longer periods of time in order to keep productivity high and reduce or avoid resource management overhead. Human resources are not only a question of numbers, but also of skills. When the product manager can foresee that certain skills will be needed to implement the product strategy, these skills can be temporarily sourced externally, or can be hired as new internal employees, or existing employees can be educated and trained.

If a software product has a very long life people retire or leave the organization for other reasons. It is part of the product manager's life cycle responsibility to keep an eye on the continuous availability of skills needed to keep the product viable even if the direct management responsibility for this is with other units, e.g. the development manager.

5.5 Market Analysis

5.5.1 Overview

The goal of market analysis is to determine the characteristics of both current and future markets, researching customers, competitors, relevant technologies and economic developments.

Organizations evaluate the attractiveness of a future market by gaining an understanding of evolving opportunities and threats as they relate to that organization's own strengths and weaknesses.

5.5.2 Concept

It is of utmost importance for a software organization to have deep insight into trends and developments in relevant markets: the markets it plays in, markets the organization wants to enter, other markets where new competitors might come from, and newly emerging markets or market segments.

Market analysis is typically performed on all three levels we discuss in this section: on the corporate level, the portfolio level, and the individual product level. Unless the company has a dedicated market analysis unit, software product managers are responsible for the product-level analysis and provide their results as input into portfolio or corporate strategy.

To conduct a market analysis, software product managers or market research specialists will typically look into the following research areas:

- Market Forces
  - Market issues: Identify key issues driving and transforming your market.
  - Market segments: Identify major market segments, describe their attractiveness and seek to spot new segments.
  - Needs and demands: Outline market needs and describe how well they are served.
  - Willingness to pay: Identify and describe for which features customers are willing to pay.
  - Switching cost: Describe the cost factors customers are facing when they switch to a competitive product.
- Industry Forces
  - Competitors (Incumbents): Identify incumbent competitors and their relative strengths.
  - New entrants (Insurgents): Identify new, insurgent players and determine whether they compete with a business model different from yours.
  - Pricing: Identify price structures and levels prevalent in the selected market segments.
- Key Trends
  - Technology trends: Identify technology trends that can threaten your business or enable it to evolve and improve.
  - Regulatory trends: Describe regulatory trends that may influence your business.
- Quantitative data about the market to support the qualitative analysis
  - Market size.
  - Competitor's revenue, profit, market share (analysis of the annual reports, if available).

When collecting the information for market analysis, the following information sources can be used:

Primary Research A fancy word for doing their own research, using

- Direct contacts inside the organization, for example colleagues from marketing, sales, support, other services, and from development.
- Direct contacts outside the organization, including ecosystem participants such as partners or media contacts, as well as customers.
  Customers may also provide valuable insights into competition.
- Systematic industry studies, from running a survey to commissioning a custom study with an industry analyst or market research agency.

Secondary Research This means using research done by others, for example industry analysts or market research agencies.

Industry analysts play an important role in IT markets: they are a valuable source of quantitative information, for example current market (segment) sizes, growth rates, market shares. They also provide qualitative information, for example market segmentation, technology and business trends, and newly emerging opportunities.

Internal Market Research Department Organizations often have specialized market research departments that act as internal service units, which can and should be leveraged by product managers. These departments conduct their own research and collect, evaluate and aggregate information from industry analysts and market research agencies, and provide regular updates to their internal audiences, in particular to product managers. Often, they also control access to the services of industry analysts. If no such market research department is available, for example in smaller organizations, software product managers may need to perform the market analysis completely on their own.

For competitive analysis we need to go beyond simple feature-by-feature comparisons of existing products. We also need to understand the strategies of competitors, their product and portfolio strategies, and their vision. To obtain this information, all the information sources listed before can be used. In addition, the website, documents, and events of competitors will provide relevant information, for example their annual or quarterly reports, materials for investors, product brochures etc.

5.5.3 Examples and Variations

Defining the Addressable Market

Defining the market that is addressed by a product is central for market analysis and helps to better understand customers. Several different segmentation models are available, one of them is the Three Level Model proposed by Weinstein in [Weinst04] (Fig. 5.6).

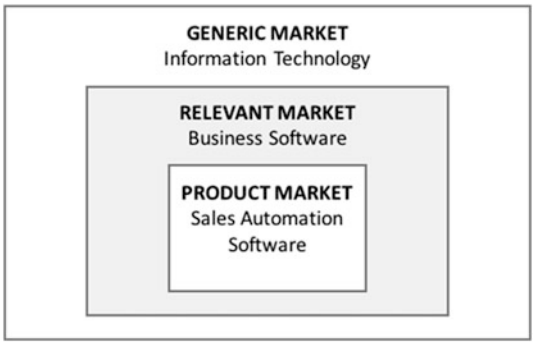

Fig. 5.6 Example for definition of level 1—relevant market: achieving balance between going too narrow nor too broad

- **Level 1—Relevant Market**
  - Define Geographic Trade Area = current market served.
  - Define Product Market = current products offered (myopia).
  - Define Generic Market = mass marketing definition (mass market).
  - Relevant Market = Larger than Product Market/Smaller than Generic Market.
- **Level 2—Defined Market**
  - Defined Market = Relevant Market segmented into penetrated market (existing customers) and untapped market (non-customers).
- **Level 3—Target Markets**
  - Apply Segmentation Dimensions to Defined Market.
  - Identify Multiple Segments within Defined Market.
  - Select Attractive Segments within Defined Market.

A key benefit of this model is the balance that can be achieved between myopia (too narrow segment definitions) and mass market (too broad definition).

Industry Analysts

Industry analysts are a valuable source not only for quantitative market information, such as size of market segments and market share of players within the segment, they also provide qualitative information, for example technology and business trends, changes in market segmentation, and newly emerging opportunities.

There are a lot of smaller boutique analysts that specialize in certain geographic or functional segments. The worldwide leaders conduct qualitative and quantitative analyses on a larger scale, such as IDC (www.idc.com), Gartner (www.gartner.com), and Forrester Research (www.forrester.com). Analysts have different strengths that need to be considered when selecting.

What they all have in common is the relatively high prices that they charge for the use of their research results. Their primary target group are usually corporate IT organizations who use the research results as input for investment decisions. Industry analysts stress their independence, although, in fact, they are forced to cooperate with the software vendors to obtain the information they need. In

addition, over time, analysts expanded their business models to include consulting, a service regularly used by vendors and corporate IT organizations alike which can easily lead to a conflict of interests. They also sell research results to vendors who want to use them for marketing purposes.

The results provided by the market research companies are nevertheless a useful source of information, even if one should not rely on them unquestioningly. It should always be remembered that market research companies do not consider it their job to merely penetrate the vendors' marketing hype and conduct serious analyses, but that they also like to produce their own hype to promote their business. In the end, product managers need to use their own sense of judgment and make business assessments and decisions in consultation with colleagues and superiors and their company-internal market research department (if available).

The fact that market research results are not only input for product management, but can be useful for marketing too, was shown by CRM software producer Siebel (in the meantime acquired by Oracle) in the spring of 2003 when it published the results of a CRM market analysis conducted by Gartner in full-page advertisements worldwide. The advertisement displayed, among other things, the "Magic Quadrant," a Gartner evaluation of companies and their products based on a system of coordinates with a "completeness of vision" axis and an "ability to execute" axis. There is no better advertisement for a vendor than to be located in the leaders quadrant, as Siebel was in the majority of the analyzed CRM segments in the above example. In the meantime, IT industry analysts impose very strict limitations on which kind of information can be used in which context: for example, they may allow a software company to use approved quotes in their marketing materials or refer to their position in the magic quadrant. But using the full magic quadrant in ads is usually no longer permitted by Gartner.

Figure 5.7 shows the Magic Quadrant's skeleton, in which companies are positioned. Gartner describes in [Hawkins08] how a magic quadrant is to be read. The axis "Ability to Execute" summarizes factors such as the vendor's financial viability, market responsiveness, product development, sales channels and customer base. The axis "Completeness of Vision" reflects the vendor's innovation, whether the vendor drives or follows the market, and if the vendor's view of how the market will develop matches Gartner's perspective. Figure 5.7 also shows how the individual quadrants should be interpreted (Fig. 5.8).

Gartner also regularly publishes the Gartner Hype Cycles, another qualitative market analysis tool that is quite influential in the IT industry. A Gartner Hype Cycle describes the response to new technologies. Gartner defines the terms used as follows (see [FennRask08]):

- **Technology trigger**: A breakthrough, a public demonstration, a product launch or some other event generates significant press or industry interest.
- **Peak of inflated expectations**: During this phase of overenthusiasm and unrealistic projections, a flurry of well-publicized activity by technology leaders results in some successes, but more failures, as the technology is pushed to its

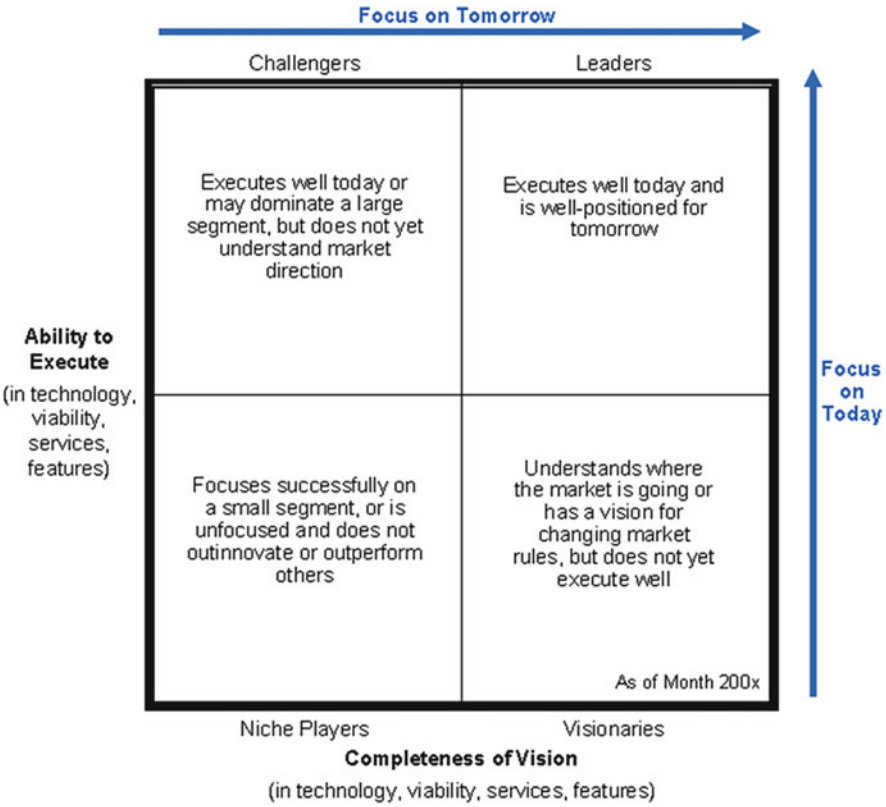

Fig. 5.7 Gartner magic quadrant (© Gartner, Inc. 2008)

limits. The only companies making money are conference organizers and magazine publishers.

- **Trough of Disillusionment**: Because the technology does not live up to its overinflated expectations, it rapidly becomes unfashionable. Media interest wanes, except for a few cautionary tales.
- **Slope of Enlightenment**: Focused experimentation and solid hard work by an increasingly diverse range of organizations lead to a true understanding of the technology's applicability, risks and benefits. Commercial off-the-shelf methodologies and tools ease the development process.
- **Plateau of Productivity**: The real-world benefits of the technology are demonstrated and accepted. Growing numbers of organizations feel comfortable with the reduced levels of risk, and the rapid growth phase of adoption begins.

New technologies positioned on the Hype Cycle do not move at a uniform speed through the cycle. When discussing a new technology, Gartner also provides their estimate how long it will take the technology to reach the plateau of productivity.

Fig. 5.8 Gartner Hype cycle (© Gartner, Inc. 2008)

This is important information that product managers need to consider in their planning activities.

5.5.4 Outcome and Inputs

A large part of market analysis results are graphics, such as market size and market share diagrams, visualization of trends and their impact on markets and on the competitive landscape. Therefore, market analysis if often documented in slide decks, accompanied by spreadsheets providing more detailed numbers and the foundation for charts.

Market analysis results are used in a number of activities of software product management, in particular product positioning, business aspects, ecosystem management, and roadmapping.

5.5.5 Summary and Conclusions

The goal of market analysis is to determine the characteristics of both current and future markets, researching customers, competitors, relevant technologies and economic developments.

Sources for market research can be classified into primary research, secondary research, and the company-internal market research department (if available).

Market analysis also includes competitive analysis. Here, it is important to go beyond simple feature-by-feature comparisons of existing products and to understand the strategies of competitors, their product and portfolio strategies, and their vision.

Industry analysts play a very important role in the IT industry, providing both quantitative market research data, as well as competitor information and qualitative insights into market and technology trends.

5.6 Product Analysis

5.6.1 Overview

Across all industries, businesses tend to become more and more data-driven. In Product Analysis, the data relevant for the management of a product is defined, located or generated, reliably and regularly accessed, aggregated based on agreed-upon definitions, and made available in an appropriate way to everybody who has a need to know. It is typically used by product managers for performance management (see Sect. 3.13) and product life cycle management (see Sect. 4.4), also by executive management as input to portfolio management (see Sect. 5.2) and for operational business management.

5.6.2 Concept

With more and more data being available to companies, it is becoming increasingly challenging to ensure that data is provided to decision makers in a way that increases the quality of decisions. For product-related data, this is what product analysis is about. We differentiate hard measures, a.k.a. key performance indicators (KPIs), and softer measures.

KPIs can be defined in four different areas. Here are often used examples:

- Financial KPIs focusing on the history, current state and plan for:
  - **Cost** of the product (development, maintenance and support or third party license fees, patent license fees). This information usually comes from the finance and controlling organization.
  - **Revenue** as well as the existing pipeline of potential customers is analyzed (license, subscription, maintenance and support revenue). This information should come from the sales and finance and controlling organizations. It can be provided per time period, per product version, etc.
  - **Profitability**, for which product-related cost are subtracted from product-related revenue.

Accounting rules apply to all the financial numbers which means that the numbers from the books may not fully reflect the actual situation. So internally, it makes sense to also look at contracted revenue.

- Customer-related KPIs focusing on the history, current state and plan for:
  - The **number of licenses** ordered, installed, new, total etc. (for a licensed product).
  - The absolute **number of active customers and end users** including growth rates and market shares (from Market Analysis) (for SaaS, internet platforms etc.). This can also be used for analyzing customer retention in the later stages of the product's life cycle. The definition of what is an active customer can be highly political.
  - The **maintenance** situation in terms of total number of customers, number of releases in maintenance und number of customers per release. This information usually comes from the support organization.
  - The **quality** situation in terms of number of support incidents and customer escalations per release. This information usually comes from the support organization.
  - **Customer satisfaction** can be evaluated through some metrics or based on qualitative analysis. Often companies conduct customer surveys on a frequent basis. Customer satisfaction is difficult to quantify. It is generally not determined on the basis of a single factor, but rather as a group of up to 20 variables regarding a range of topics, such as reliability, documentation, usability, service quality, sales coverage, etc. [JohGus00, Myers00].
- Development-related KPIs focusing on history, current state and plan for:
  - **Quality** during the development process and its relationship to customer-perceived quality (see above).
  - **Productivity** of the development team.

Some development organizations tend to consider this data as internal, but a product manager needs to look at this data at least on a summary level.

- Product-usage-related information and KPIs:
  - For licensed software products where the runtime environment is under the responsibility of the customer, runtime measurement is usually limited and may require the customer's agreement.
  - In an internet environment like Software-as-a-Service (SaaS) or a (self-developed) e-commerce platform, measurement is easier, since software and runtime environment are both under the same company's responsibility. This includes **web analytics** measuring, click rates and analyzing visitors as well as detailed monitoring how users interact with the software [CrolYosk13]. Some internet companies use customer discovery to test the user acceptance of new features.

This data can be helpful for a product manager's decisions regarding requirements (see Sect. 4.2).

All the numbers need to be considered over longer periods of time, i.e. not only current actuals, but also in comparison to the history, the plan and budget, and possibly the market (with data from market analysis, see Sect. 5.5).

Customer satisfaction can be considered as one of the softer measures. They also include qualitative input like feedback from market analysts, trade press articles, individual customer feedback, information from the sales channels (like win/loss analysis or opportunities), information from the support and service functions and more.

5.6.3 Implementation

When an organization has more than a few products in its portfolio, we recommend to work with common definitions for the selected measures so that numbers are comparable. Then productivity can be optimized by establishing a central data analyst or team of data analysts who collect the relevant data reliably and regularly, aggregate them, and make them available in a way that helps the decision makers, be it product managers or executive managers. Standardized graphical representations can add value here.

When there is no central product analysis, this is part of the responsibility of each product manager or product management team. In that case, the product manager can focus directly on those measures that are selected and relevant for a particular product. This selection can change dependent on the product life cycle phase (Fig. 5.9):

Even if there is a central data analyst responsible for product analysis, it is important for product managers to fully understand how measures are defined, where the data comes from, and how the numbers are aggregated. This may require some deep-diving. When done for the first time, results are often surprising, like product revenue that is accounted as service revenue (from combined product-

Phase	Focus of product analysis
Conception and creation	Financial KPIs for planned data, Development-related KPIs
Market introduction	Product-usage-related KPIs based on planned and current data, Financial KPIs for planned and current data, Development-related KPIs
Growth	Customer-related KPIs, Financial KPIs, Product-usage-related KPIs, Development-related KPIs
Maturity	Financial KPIs for current data, Product-usage-related KPIs, Customer-related KPIs, Development-related KPIs
Decline	Customer-related KPIs, Financial KPIs, Development-related KPIs
Withdrawal	Financial KPIs for historic and current data

Fig. 5.9 Key Performance Indicators (KPIs) in product life cycle phases

service deals), or cost accounted against the product that has absolutely no relationship to the product. Those findings require correction.

The product analysis results are used in a number of activities of software product management, in particular business aspects, performance management, life cycle management, roadmapping, release planning, and product requirements engineering.

5.6.4 Summary and Conclusions

For a software product manager, it is of key importance to have reliable product-related data updated frequently as a basis for decision making. Company-wide standards regarding the selection, definition, data sources, aggregations and graphical representations help the understanding and comparability between products significantly.

5.7 Corporate Strategy Processes

On the corporate and/or business unit level, strategy processes are usually governed by a yearly calendar that ensures that business planning is finished in time for a new financial year. The financial year may not be identical to the calendar year, e.g. Apple's financial year starts on October 1. At the beginning of the financial year, all stakeholders within the organization need to know what the objectives, allocated resources and budgets are within which they are supposed to work. Corporate business planning needs to be aligned with an updated corporate strategy which in turn needs to be aligned with updated product strategies. So all this update work can be scheduled on the yearly calendar as well (see also Sects. 3.14 and 4.5).

Documentation of a corporate or business unit strategy can differ significantly from company to company. Some companies go through the annual update cycle with great consequence, in particular publicly traded companies. They usually document the results internally, and publish a subset externally. A documented corporate strategy is very helpful for product managers since they can use the corporate strategy for alignment and also for justification of their respective product strategy.

Unfortunately, there is a surprisingly high number of companies that do not update their corporate strategy on a yearly basis. There may be no need for such an update frequency—that is rare for software—, executive management may be hesitant to document a strategy because they are unsure where they want to go, or there are political reasons for not documenting the corporate strategy. The latter happens frequently in companies where owners and customers are partly identical with corresponding conflicts of interest that the executive management does not want to address explicitly. All these situations make life more difficult for a product manager.

Orchestration of the Organization's Functional Areas

6

It is part of the concept of an "enterprise" that people with different abilities, experiences, and skills work together as employees to reach common goals. Typically this cooperation is governed by a division of work so that the individual strengths of each employee can be optimally utilized. The task of management lies in the definition and communication of the goals—strategic to operational—to create organizational structures that lead and support the cooperation, to check the progress towards the goals frequently and to intervene whenever necessary. In this sense, software product management is a comprehensive management task to its full extent focused on one or several software products.

The division of work for a software company usually means that there are separate functional units. The ISPMA SPM Framework identifies four key functional areas that a software product manager usually needs to orchestrate: Development, Marketing, Sales and Distribution, and Services and Support. The framework lists the core tasks that each of these functions performs in the respective columns of the framework. In addition to these four key areas, orchestration of other areas is often required, in particular, the SPM units of other products within the company, Finance, Controlling, Human Resources (HR), Legal, and Research. Here, the product manager's orchestration responsibility includes the alignment of product strategies and plans, research and innovation initiatives for both functional and technical innovations, resource management, and availability of correct and timely measurements.

Corporate IT organizations more and more are organizing themselves like software companies, often without differentiating between Sales and Marketing. Here the existence of HR and Legal depends on the degree of independence from the parent company.

Whether Software Product Management is a separate organizational unit or not, product managers collaborate with a wide range of stakeholders from both outside and inside their organization. Inside the organization, they work together with other business functions in a cross-functional product team that ideally stays in place for the long term, i.e., not merely for a single release. The purpose of this team is ". . . to

H.-B. Kittlaus, S.A. Fricker, *Software Product Management*,
DOI 10.1007/978-3-642-55140-6_6

manage all the elements needed to achieve the financial, market, and strategic objectives of the product as a business" [Haines14].

The cross-functional product team has members from different departments in the organization and may often be virtual, with members from different geographic locations or cultural backgrounds. In this setup, product managers are challenged to achieve "influence without authority, accountability without control" ([Hall13], p. 1). This task is conflict-laden in several dimensions.

Orchestration challenges typically exist:

- Between product managers for different products in the organization's portfolio due to resource allocation, budget allocation, positioning of products against each other within the portfolio, functional scope, interfaces, release dates, and bigger requirements that require parts of the implementation in different products with resulting dependencies,
- Between sales and development unit due to short-term goals and individual customers versus long-term abstract perspectives,
- Between product manager and development unit due to release scope and dates, resource allocation, and project planning (overall project portfolio, not details in each project),
- Between product manager and marketing unit due to brand marketing versus product marketing, and
- Between product manager and sales and distribution unit due to discounting, the reliability of commitments regarding the sales volume of the product, reporting granularity, customer commitments and what can be delivered realistically versus everything that customers want.

Units like Sales and Development differ in their cultures in all industries, but nowhere as severely as in software. While sales people usually have an orientation towards short-term goals and individual customers, developers tend to have a long-term abstract perspective. Denning and Dunham state in [DenDun03], p. 141: "Information technologists are trained to view worldly objects as potential abstractions to represent inside a computation. Customers collapse into abstractions in this world. Abstractions do not have concerns or make assessments." On the other hand, salespeople tend to make a sale at almost any price even when customer requirements have to be committed that endanger the focus and consistency of a software product.

Condon tells in his book [Condon02] how the software product manager has to act as a mediator in conflicts like this one. Very rarely has a software product manager the right to give orders to all relevant organizational units. Typically he needs to sell and persuade. Such persuasion may be easy if he is considered competent in the subject under discussion. But since nobody can be equally competent in all areas, leadership in a non-hierarchical sense is required. However, often the product manager does not act as leader, but as Jack of all trades. In those companies, executive management, as well as the managers of the other organizational units, tend to misuse the product manager as caretaker for any problem that

comes up and does not clearly belong to one of the units. As a consequence, the product manager is forced to give higher priority to the urgent tactical problems than to the important strategic ones which often means that he cannot deal with the important ones that influence the mid to long term success of his products at all. Executive management is not well advised to allow such a situation. What can help are clear mandates that include the following:

- Clear definition and separations of concerns, responsibilities and competencies, and clear definition of accountability,
- Delineation of responsibility between product managers and project managers,
- Definition of release planning, requirements management, and quality assurance processes, and
- Establishment of clear channels of communication between product manager and different organizational parts (for example sales).

In addition to those mandates, roadmaps are also an important tool that supports collaboration with the various company functions.

To achieve excellence in orchestration, software product managers need to drive process improvement in working with other functions. Excellent software product managers are not just reacting to operational challenges. Instead, they pro-actively foster better relationships and drive better processes for collaborating with these other functions.

In this chapter, we look in more detail at the orchestration tasks towards Development, Marketing, Sales and Distribution, and Services and Support and then describe the associated skills requirements.

6.1 Role and Processes

The role of orchestrator is one of the central tasks of SPM. It aims at optimizing the cooperation of all units for achieving product-related goals. Each unit is expected to contribute to product success in the best possible way. Conflicts of interest are bound to exist among the units and sometimes even within the product management organization. This task calls for successful negotiations in case of conflicts of interest [FrGoBySc10, Thomp14].

Conflicts may surface when product strategy decisions are to be implemented. For example, a change in pricing that affects the pricing metric may impact development to implement changes in license reporting or enforcement, support to update support pricing, marketing to update the price list, license key generation, and to communicate the change to customers, and sales needs to be trained as well.

Conflicts may also surface in operational tasks. For example, resolving an escalated customer problem may require the product manager to change a roadmap to provide a good longer-term solution. He may also need to coordinate activities between support, development, and sales (account management for the customer) to find an acceptable workaround for the short term.

The SPM Framework can be used to know which activities of other functions—represented by cells in the orchestration columns—are affected by the decision or problem at hand. This knowledge can help identify the stakeholders that need to be involved.

Once the right stakeholders have been identified, the software product manager needs to use their understanding of the other functional units to map out a suitable approach for addressing the situation. Such an approach may include establishing role clarity and defining a process to use to address the situation at hand. Then, the product manager can exercise his leadership skills to work through the agreed process with the stakeholders.

Software product managers typically don't have the power to direct the other members of the cross-functional team. To achieve a constructive, successful collaboration with these other functions, software product managers can use two levers: First, they need to exercise their leadership skills (see Sect. 6.6). Second, they need to achieve a common understanding of roles, responsibilities, and contributions of each member of the cross-functional product team.

For setting expectations and achieving role clarity, it is ideal to have a comprehensive process model. Process models can be characterized by

- Their scope regarding the business functions they cover and
- Whether they explicitly recognize different stages of the product life cycle.

The ideal situation from an orchestration perspective is an up-to-date process model. That process model should take a holistic product view and cover all business functions involved in taking a software product to market. Also, the model should be aware of life cycle stages and cover the entire life cycle of a software product from inception to withdrawal, i.e. be a product development life cycle (PDLC) model. The coverage of the life cycle is important because the different stages of a product's life cycle require vastly different contributions from the various business functions.

Unfortunately, few software organizations have implemented a comprehensive process model. Instead, many process models are established that only cover development activities—a software development life cycle (SDLC). These types of process models are often based on the chosen software development methodology, Scrum for example, and do not distinguish between different stages of the product life cycle. SDLC models may be helpful for clarifying roles and responsibilities when collaborating with the development team, but from a software product manager's perspective, they are incomplete since they do not address other critical functions that are required to take a software product to market.

In all cases where the product manager cannot lean on an up-to-date, comprehensive PDLC, product managers need to negotiate rules of engagement, deliverables, and timelines individually with the other business functions represented on the cross-functional product team.

A RACI matrix [Haines14] is a frequently used tool to clarify roles and responsibilities. A RACI matrix defines who is Responsible, Accountable,

Consulted, and Informed for each activity or deliverable within a process. Based on the RACI matrix, functional support plans (FSPs) can be established: "FSPs are created as action planning, horizontal contracts across the cross-functional product team's memberships. FSPs describe the activities, deliverables, dependencies, and schedules for each team member across the entire life cycle . . ." [Haines14]. In most companies today, this is handled in a more informal way.

In the long-term, it makes sense for software product managers to push for the development and introduction of a suitable PDLC. The PDLC will reduce the effort involved in negotiating FSPs with multiple different business functions.

6.2 Development and UX Design

The development unit is responsible for all technical software aspects including the implementation of extensions and changes to the software. The development function exercises a strong influence on the product's functional capabilities and qualities, as well as the user experience. Therefore, successful collaboration with Development and User Experience (UX) Design is a key success factor for software product managers—and there are many possible areas for conflict.

The Development column in the SPM Framework details the tasks of Development.

Engineering management addresses all development aspects that are relevant across and above development projects. These include governance of product architecture, development processes and tools, configuration management, knowledge management, resource and skills management, development sourcing, and estimations.

Project management addresses the execution of a release plan in software development. The execution is typically done based on a project organization. The project is also responsible for writing internal documentation and contributing to the software-related external documentation.

For a project, an appropriate development methodology must be selected and followed. This choice has an impact on the way SPM and the development project cooperate. Development organizations use a variety of methodologies. The chosen methodology on the development side has an impact on the work of the software product manager and the interface between SPM and Development—how requirements are handed over for implementation and how the acceptance of project deliverables is managed. Most companies use a mix of different methodologies, be it agile, iterative, or stage-gate, often called waterfall. Popular agile and lean methodologies include Scrum, Kanban, and eXtreme Programming (XP). When agile methodologies are implemented in their pure form, there may not be a project organization anymore. In that case, we use the term "Development Management" instead of "Project Management".

At a high level, agile as well as iterative development are driven with small, controllable steps or iterations. With most agile methodologies, every iteration consists of analysis, design, coding, and testing. With iterative, only coding and

testing are done in iterations while analysis and design are done upfront. With stage-gate, or waterfall, there are no iterations, but one stage or phase is done after the other. So, with stage-gate and iterative, requirements are handled early whereas with agile, requirements usually need to be handled in each iteration.

There is a role in agile projects that deals with requirements. In Scrum, that role is the **Product Owner**. In smaller agile projects, the SPM may assume the Product Owner role, but that does not scale up [Kittlaus12]. In larger projects, the Product Owner role needs to be filled with additional team members who cooperate tightly with the SPM. The Scaled Agile Framework (SAFe, [Leffing16]) offers an approach for scaling agile in larger organizations.

With agile methodology and continuous integration (and even continuous deployment), new functionality can be delivered daily if customers can deal with such a high release frequency (see Sect. 4.3). The SPM challenge is to focus on the strategic and important items in such a high-pace environment.

Project requirements engineering is part of the project team responsibility and follows a process like the product requirements engineering process (see Sect. 4.2). Once the contents of a release are defined, the corresponding product requirements are transferred into project requirements and further refined. Project requirements can also address development-internal needs that are raised in the course of a project. Tracking the project requirements and synchronization with the product requirements are continuously required. With agile methodologies like Scrum, a large part of the requirements analysis is done by the product owner. In that case, we use the term "Development requirements engineering" instead of "Project requirements engineering".

User experience (UX) design addresses every aspect of the users' interactions with a software product or component (see Sect. 6.2.2). UX may be part of Development's responsibilities or organized as a separate shared-service organization. The UX design scope and expectation for a product are described in the Product Strategy under Product Positioning (see Sect. 3.6).

Quality Management addresses the technical quality of software. It includes test concepts and infrastructure, technical support concepts and structure (together with the Support unit), a historical quality database, quality forecasting, and the execution of tests.

The product manager's orchestration responsibility includes the acceptance of results based on tests, agreements on release scope, schedules, and estimates, tracking of the execution of plans, tracking of the project vs. product requirements, and negotiations and adjustments of plans including scope changes if needed.

6.2.1 Organizational Setup, Roles, and Processes in Development

The organizational structure, job titles, and processes used by the development function depend on several factors. These factors include the type of software product and the development method, e.g. waterfall, iterative, or agile, and the process chain from code development to product delivery.

The typical setup in a traditional software development organization using a waterfall or iterative development process includes the following roles:

- Development team, including developers, architects, and development managers
  This team may be relatively stable with developers working on the same product for many years. In that case, the development team has deep and highly relevant product knowledge;
- Separate QA team—by intent, members of these teams often are different from the developers.
- Engineering support teams: e.g. roles for maintaining development and test environments, configuration management, etc.

The setup of an agile development organization that uses Scrum, for example, consists of the following roles [SuthSchw13]:

- Product Owner
  Software product manager and product owner are two different roles that need to be closely aligned. The same person may assume both roles in smaller organizations. Often, however, the combined workload is too high for one person, and the job profiles are hard to reconcile: e.g. SPMs often need to travel quite a bit, while product owners should always be available to the product team.
- Development Team(s) of fewer than ten people each
  In contrast to the typical setup for waterfall development, testers in agile development are part of the development team.
- The Scrum Master who is "[. . .] responsible for ensuring Scrum is understood and enacted [. . .] by ensuring that the Scrum Team adheres to Scrum theory, practices, and rules."

A software organization may introduce additional coordination layers and functions to scale the development organization beyond a small number of Scrum teams. For example, a release management function may be established, as suggested by the Scaled Agile Framework (SAFe, [Leffing16]).

Some agile gurus tend to be dogmatic arguing that the role of product owner covers everything that is needed for product management and the role of SPM is superfluous. We disagree. In our experience, the operational character of the product owner role does not leave enough time for the strategic tasks of product management. Therefore, a separate SPM role is needed. Our view is also described in SAFe [Leffing16]. If in a smaller environment, it is possible to assign both the SPM and the product owner role to the same person, this person needs to cover all SPM tasks as described in the SPM Framework and this book.

Continuous Everything and DevOps

Software organizations increasingly automate the processes that lead from finalized code changes to the deployment of new software in production environments.

The software product manager needs to be aware of the technical capabilities of the development organization and needs to make sure that the actual delivery to customers is aligned with business needs: how to use these capabilities must be a business decision, based on what the market can absorb.

The processes that lead from finalized code changes to deployment in production environments include:

- Continuous Integration: automated integration, build, and test in the development environment,
- Continuous Deployment: automated push of software into the production environment, and
- Continuous Delivery: automated delivery of software to customers.

If the continuous integration, deployment, and delivery capabilities are in place, the organization can choose how frequently to deliver code changes to customers. Continuous deployment can result in multiple production deployments per day, for example in a SaaS delivery model [HumFar10].

To implement continuous deployment and delivery, Development needs to collaborate closely with other functions. If the software vendor also operates the software, for example in a Software as a Service (SaaS) delivery model, tight integration with the Operations function is required. This integration is called DevOps: "DevOps is about aligning the incentives of everybody involved in delivering software, with a particular emphasis on developers, testers, and operations personnel" [HumMol11].

In a DevOps context, it can be beneficial to establish a "release heartbeat" where new functionality is released to the market at regular intervals, e.g. once every week or every 6 months. The heartbeat drives alignment and efficiency within the software organization and properly sets expectations of the market.

Typical Areas of Conflict: Development

Typical areas of conflict between SW Product Management and Development include:

Release Planning: Disagreement About the Priority of Requirements

The disagreements may be related to individual requirements or features, or to classes of requirements: for example, SPMs often focus heavily on functional requirements, while Development may push to spend more effort on architectural improvements, code cleanups, reduction of technical debt, or on addressing non-functional requirements.

Release Planning: Development "Does What They Want" Instead of Executing the Release Plan

For example, Development may consider the list of requirements to incorporate in next release or the planned release date as unrealistic considering existing resource

and schedule constraints—and simply goes off to execute based on priorities they set themselves.

Requirements Engineering: Disagreement About the Meaning of Requirements
The software product manager does not agree with the way Development has translated certain product requirements into project requirements.

Requirements Management: Changing Requirements
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. From a business perspective, however, changes in the market, in customer situations, or on the legal side can lead to changes in project requirements at any time. In these situations, it is the task of Development's project requirements management in cooperation with SPM to ensure timely decisions. The additional efforts in development that are caused by such changes depend to a certain degree on the type of development process employed. The waterfall model assumes a strictly sequential process and cannot cope well with late changes. Iterative or Agile models are better suited to handling change. Also, iterative and agile models provide early indicators of project success and customer acceptance of the results. They allow in a relatively early phase and continuously throughout the project to get hands-on experience with a prototype.

Agile Development: Misalignment of Sprints with Product Strategy
Over multiple short sprints, the product increasingly evolves into a direction that is not in line with the release themes and epics defined by the product manager.

Continuous Deployment: Disagreement About Deployment Frequency
Once continuous deployment capabilities have been set up, Development and Operations may want to use these capabilities to a full extent. However, the software product manager may want to exercise more control over what is deployed when and exert power to group changes together or to delay deployment of certain changes.

It can be helpful to give Product Management the role of the internal client to ease or avoid these conflicts. The client role includes responsibility for budgets, contents, and acceptance of the results of the development activities. Budgeting needs to be ensured in the planning processes of the company (see Sect. 5.2). The content is managed in the release planning and requirements management processes (see Sects. 4.2 and 4.3). Acceptance of development results is based on reviews and tests that are part of the development process. The Software Product Manager must rely on qualified internal or external reviewers and testers, e.g. a Quality Management unit, who provide a reliable basis for his acceptance decision.

A strict separation of client and contractor is usually somewhat difficult to start for Development since the developers tend to see this more as an annoying control than as help whereas people on the business side often feel forced into sharing

responsibility for the results of development. If these initial difficulties can be overcome, the separation usually turns out to be helpful. It makes the role definitions more precise and leads to better cooperation. The separation does not prevent conflicts between Product Management and Development, however. It simply clarifies the rules of conflict resolution.

An ideal of Software Engineering is expressed in the Capability Maturity Model Integration (CMMI) of Carnegie Mellon University. It states that the "perfect" software organization can work with all types of development processes in the same reliable and masterful way and selects the optimal type at the beginning of each project following certain criteria. Few development organizations are perfect in that sense. Most are glad when they master one process type in a reliable way. We do not recommend the Software Product Manager to enforce any process type that the development team does not master. An unsuitable process would increase the probability of project failure. Rather, the product manager should motivate Development to adopt more flexible process types depending on the type of software. For example, the development process for software games is highly prototype-oriented and seems to be closer to the production of a Hollywood movie than to traditional commercial software development [Waldo08].

The software product manager is dependent on being able to predict if Development can meet the plan in terms of time and quality. Once Development has written the project plan, the product manager is advised to follow the progress of the project in comparison to this project plan. He must ensure that marketing and sales are ready for the launch of a new product or release. Mistakes in the prediction can result in significant cost, bad press, and revenue loss. Since these mistakes happen within most software organizations, a lot of software companies do not publish fixed dates. Instead, they start with limited availability for pre-selected customers before the product becomes widely available. Internal IT organizations may follow the same model by introducing new applications for a small number of users first. However, this approach is often impractical for company-wide applications without building extensive interface code—e.g. payroll. With agile methodologies, often part of the defined release contents is sacrificed in order to meet the planned date.

6.2.2 User Experience (UX) Design

User Experience (UX) Design can be anchored at different positions in the org chart, but often resides within or close to the development function. UX has a strong impact on product management and marketing, support, and development.

UX is a broad term covering and interacting with disciplines like graphic design, information architecture, Human-Computer-Interface (HCI) design, interaction design, and usability engineering [ShPlCoJa13]. UX addresses every aspect of the users' interactions with a software product or component with the purpose of shaping the user's behaviors, attitudes, and emotions about that product or component. Emotions include delight and annoyance about the product [LeCBölPe13], excitement and fear in games, and a feeling of being in control when using decision-support software. UX must take human-system interaction, user interfaces, device

and workplace ergonomics, service and content offered by the product, the context of product use, and standards like ISO 9241 [ISO98] into consideration. UX also needs to consider the various standards and user expectations, which may vary widely across types of software and market segments.

Cagan distinguishes four UX design-related activities that are critical to the success of software products: interaction design, visual design, rapid prototyping, and usability testing. These four roles need to "[. . .] work closely with the software product manager to discover the blend of requirements and design that meet the needs of the user." [Cagan08].

Startup communities have developed a variant of UX, Lean UX, to "[. . .] break the stalemate between the speed of Agile and the need for design in the product-development life cycle." [GotSei13]. Lean UX is based on three foundations: design thinking, agile software development, and the Lean Startup method. All three methods emphasize experimentation, rapid iterations, and deep customer and user involvement. The principles and techniques of these underlying methods are then applied to the design process. "[. . .] this is the essence of the Lean UX approach. Only design what you need. Deliver it quickly. Create enough customer contact to get meaningful feedback fast." [GotSei13].

Typical Areas of Conflict: UX Design
Due to the objectives of UX Design, there is a significant overlap with the product manager role, especially in the following areas:

- Developing a deep understanding of customers' true needs.
- Understanding intended product usage.
- Developing product scope and product definition.
- Eliciting product requirements.

In these areas, software product managers may find that UX designers are powerful allies that help them define a product that serves customers and users even better. Alternatively, they might be in stark conflict, quarreling over decisions and accountabilities.

A software product manager is well advised to canalize the creativity of UX designers into the refinement of early product concepts and utilize their experimentation skills to get evidence that the product concept works for the intended users. If the UX designers discover significant problems in user acceptance and product effectiveness, the product manager may have to pivot the product concept.

6.2.3 SPM's Focus Areas for Orchestrating Development

An SPM needs to focus on the following areas when orchestrating development:

- Acceptance of results based on verification and validation tests.
- Release scope and dates, project planning.

- Execution of plans.
- Synchronization and tracking of project vs. product requirements.
- Estimates.
- Resource, knowledge, and skills management.

The less experience a development organization has with software product development, the more they will underestimate the extra effort that comes with a product vs. a software that just runs in one customer environment. This situation needs an SPM's special attention regarding estimates and the execution of plans.

6.3 Marketing

Marketing is a top priority for a software company. Successful software companies spend a high percentage of their revenue on sales and marketing—on the average twice as much as for research and development of new products. There are several reasons for this:

- Software is an intangible commodity that is not easy to describe. The intangibility makes it more important and time-consuming to give the customer an "idea" of the product by way of marketing measures.
- Due to low variable costs, gross software sales margins are very high. These high margins can be used to pay for expensive marketing measures, boost sales numbers, and thus create a yet higher profit margin, even after marketing expenses have been deducted.
- The law of increasing returns (see Sect. 2.3) in the software industry challenges every vendor to have his products achieve market leadership. Sales numbers and market share are therefore crucial for every software company.

In this book, we assume that software product management and marketing are two separate tasks that are assigned to two separate units within a company. However, in cases when a company does not have a separate marketing unit, the software product manager often must take care of some or all of the marketing responsibilities and may get the title "Product Marketing Manager" or similar.

The Marketing unit is responsible for all aspects of preparation and support of the product sales activities of a company including the positioning of the product in the market and the creation of product awareness. The actual split of responsibilities between Marketing and Sales may differ from company to company. Typically, companies establish a company-wide marketing strategy within which the marketing activities for individual products are defined and executed. The Marketing column in the SPM Framework describes the main tasks of Marketing.

Marketing planning addresses the development and negotiation of plans for all marketing-related activities during a given timeframe, often a year, including respective budgets. The plans can be product-specific, or for groups of products.

They need to be synchronized with corporate and product strategies and plans, and the sales plan.

Customer analysis means the analysis of existing or potential customers or groups of customers regarding additional business opportunities and retention. The analysis is frequently repeated.

Opportunity management means the pursuance of identified business opportunities with the objective to turn these opportunities into concrete product success. Opportunity management may include the formulation of product requirements, development and implementation of new marketing approaches, and tight cooperation with Sales. It is performed continuously.

Marketing mix optimization means the selection, implementation, and management of appropriate marketing tools. Since we have covered product and price under product strategy (see Chap. 3), here the focus is on promotion and communication, and on channels appropriate for a product, and the management of marketing partners within the product ecosystem. All require tight cooperation with SPM and synchronization with the corresponding product strategy.

Product launch means the introduction of a new product, version, or release to the market. Marketing needs to orchestrate the activities that serve to create attention of existing or potential future customers, in the trade press, with market research agencies, and so on. Typically, SPM, Development, Sales, executives, partners, and sometimes customers are involved.

Operational marketing means the execution of the marketing plan, tracking of the relevant measurements, and taking corrective actions when measurements deviate from the plan.

The product manager's orchestration responsibility includes the positioning of a product in the marketing plan, tracking of plan execution, and cooperation in product launches, channel, and partner management. The product manager may decide to participate in marketing events selectively.

The product marketing department of a software producing organization is responsible for all aspects of preparing and supporting product sales. Their goal is to craft an effective marketing and sales funnel: attracting as many prospects as possible, turning them into leads that are interested in the product, converting them into paying customers, and ultimately turning them into promoters of the product.

This product marketing goal closely aligns with essential responsibilities of a software product manager as identified by Ebert [Ebert07]: to conquer markets and grow market share. Because of this, effective collaboration with and orchestrating of the product marketing department is a core determinant for creating a winning product [GriHau96]. Nevertheless, the core activities of product management differ from those of product marketing. Product management defines the product to be built, while product marketing defines how to communicate the product to the market.

6.3.1 The Marketing Organization

The marketing of a software product has the following three concerns [KotArm15]:

- Strategic marketing: defining a strategic direction for marketing that is in accordance with the corporate strategy,
- Product marketing: translating the marketing strategy to the product level, and
- Marketing communication: executing the marketing strategies to create tangible deliverables such as promotional materials or event booths.

Depending on the size and design of your organization, these concerns can be addressed by one single person, a team, or even multiple teams. The three concerns have a reciprocal relationship. The marketing strategy influences the marketing goals that the product marketing team pursues. Similarly, the product marketing decisions impact marketing communication (MarCom). The success of the MarCom activities, then, affect the key performance indicators (KPI) of all three marketing disciplines. The MarCom KPIs inform the product KPIs, which in turn inform the strategic KPIs.

Strategic Marketing
The goal of strategic marketing is to create a recognizable brand. Together with the Board, the Chief Marketing Officer formulates, analyzes, and evaluates a marketing strategy in accordance with the corporate strategy for positioning and presenting the company or products in the market. They define a clear message that the other marketing disciplines embed in their activities. For example, upper management may make a strategic decision to focus marketing investments more on the company's brand or on specific products.

Especially for software, which is a product type that is difficult to describe and to communicate, the creation of a strong brand is essential for successful market recognition. In the software industry, the marketing expenses compared to revenue are higher than in most other industries. It can be observed that a software vendor spends 5–6% of its total revenue for external marketing programs. This spending does not include personnel expenditures and other internal expenses.

There are various qualities that characterize a successful brand, which a software product manager must be aware of:

- A brand is based on an identity that has gradually developed, namely the corporate identity, which consists of company-specific competence coupled with the experience and culture of the company.
- The core values of a brand need to be outlined as clearly as possible and easy to understand. Focusing on only a few dimensions constitutes a key success factor.
- A strong brand develops its own "ecosystem" as described in Chap. 3. Such an ecosystem is decisive in determining how a brand and a company develop, thus has a restrictive character. An ecosystem does not permit arbitrary diversification attempts, as these could be detrimental to the brand.

- Strong brands require publicly visible profiles. These may be created based on technological leadership or the fringes of a niche market and are difficult to develop in a mainstream market.
- Strong brands owe their existence to a convincing core element, namely company-specific competencies, or to their fascinating aura. The addition of an emotional desire to the brand's core creates the ideal situation. In the IT market, Apple is a good example for generating such an aura.
- Especially in difficult times or in saturated markets with intensive competition, strong brands have a clear advantage and tend to improve their market position in comparison to weaker competitors.

A source of discussions in a multi-product company is the conflict between brand marketing and product marketing. The software product manager wishes marketing for making the product known and understood by specific target groups or for boosting product sales. If marketing aims at establishing a brand, however, it will invest its limited resources in establishing and maintaining the publicly visible brand profile.

A brand is a valuable asset for a company. A product marketed under this brand name will benefit from the brand recognition. Oracle, for example, consistently markets all its products using the corporate name. This brand name is consistently used for its diversification strategy and to market both Oracle database products and other products, such as financial applications or application servers.

A company such as IBM, on the other hand, would have difficulty marketing all its products using the corporate brand name IBM. IBM offers a too wide range of products and services. The brand is still valuable, but its breadth makes it difficult to evoke the associations desired for specific software products. Particularly in the software market, where IBM is traditionally not recognized as a major player, brands need to be differentiated. IBM, therefore, continued to use the brand names "Lotus," "Tivoli," and "Rational" after the acquisition of these companies to achieve better recognition in the respective market segments and to prevent losing the intrinsic value of these brands. At the same time, immediately after acquiring Informix or Sybase, IBM merged these brand names with its own IBM DB2, which never attained the same brand status as Oracle. What we can see here is the area of conflict between the corporate brand and the individual brand names for products or product families, reflecting the conflict of interests between corporate marketing and product marketing. IBM has never succeeded in establishing "IBM Software" as a distinct brand despite attempts to do so, perhaps because these attempts were not consistently pursued or implemented.

An effective marketing strategy influences the end-results of the MarCom activities. Therefore, popular strategic KPIs include both strategic concerns and tactical result metrics:

- The **Net Promoter Score** for a brand is a method to measure customer loyalty by asking a single 0–10 scale question: "How likely is it that you would recommend

the brand to a friend or colleague?". The promoter score is the percentage of customers who answer 9 or 10 subtracted by those who answer 0–6.
- The **Brand Awareness**, typically expressed as a percentage of a target market, is a measure of whether potential customers correctly identify a brand. The measure indicates whether a customer can respond to the brand after viewing its logo or packaging.
- **Return on Marketing Investment** (ROMI)—a simple Return on Investment (ROI) formula calculated as follows: [Incremental Revenue Attributable to Marketing (€) * Contribution Margin (%)—Marketing Spending (€)]/Marketing Spending (€).

Product Marketing

Product marketing starts with market and customer analysis. Marketing opportunities are identified by analyzing the relevant market segments, understanding their respective needs, and defining the appropriate communication approach for each segment. Marketing is responsible for this process in close collaboration with product management based on the contents of the product strategy. For a new product release, Marketing prepares the release launch by preparing the appropriate marketing communication deliverables and possibly by organizing events.

The KPIs for product marketing are roughly like those of strategic marketing, but on a product level instead of brand level. The product-level KPIs need to be aligned with corporate objectives and used to inform the strategic marketing KPIs. The product-level KPIs should at least include:

- The **Net Promoter Score** for a product by asking a single 0–10 scale question: "How likely is it that you would recommend the product to a friend or colleague?". The Net Promoter Score is the percentage of customers who answer 9 or 10 subtracted by those who answer 0–6.
- The **Product Awareness** is comparable to the brand awareness measurement but also includes whether the brand is correctly associated with the product.
- **Conversion** is a measure of how many of the potential customers converted into paying customer.
- **Cost per Lead** is a measure of how much, in average, had to be invested to acquire one new lead.
- **Return on Investment** is the average revenue per lead divided by cost per lead.

Marketing Communication

Marketing communication, MarCom or operational marketing, concerns the execution of the marketing plan and results in concrete marketing outputs. Traditionally, this activity was focused on producing print media, television, and radio advertisement. Nowadays, marketing communication also includes the use of the brand language, public relations, sponsorship, social media, tradeshows, and in-product communication. Within a software producing organization, Marketing takes ownership of these activities and discusses the deliverables with the software

product manager. The software product manager should only be actively involved in marketing communication activities in exceptional situations.

The marketing communication plan for a product is derived from the marketing strategy and defines the objectives for the individual marketing measures. Besides advertising, important measures are public relations, online marketing, telemarketing, and sales support. A special case is the launch of new products, new versions, or major releases of existing products. Usually, a launch plan for each product launch is developed and implemented that should be integrated into the overall MarCom plan. The significance of launch activities depends on the type of product. Typically, the launch focused on a release date is important in the consumer market. In the enterprise market the focus on a hard release date is no longer the rule. Also, adoption of new products or versions takes a lot longer with enterprise customers than in the consumer market. Therefore, launch activities are spread over longer periods of time. There is a lot of literature about product launches like [Lawley07] or [Cooper00].

A combination of the various marketing components is used for integrated marketing in the software industry. The MarCom Plan describes media planning as well as the campaign's creative strategy for advertising and public relations online and offline. Public relations have top priority in marketing communication for software companies. Leaders of successful software companies spend a lot of time discussing market trends, corporate visions, and strategies with the technical and business press as well as with analysts.

User conferences are another important forum for a software company. The company may organize such conference for its products (e.g. SAP's SAPphire) or join industry conferences (e.g. the European Banking and Insurance Forum, EBIF, JAVAWorld, etc.). A software company needs its corporate management to be visible and approachable in such conferences and involve chief architects, community leaders, and product managers.

The chosen product development methodology will impact the marketing orchestration approach. The typical marketing department sets plans, goals, and budgets at the beginning of a new year or period and will be unwilling to deviate from these plans substantially. This resistance can create frustration in a high-pressure, quickly changing environment that is typical for a software product organization. It is good practice to make agreements with the marketing department on the type of work and how much work they can do for a product on an intermittent basis. For example, 10% of marketing resources could be made available to product management for unforeseen feature announcements.

6.3.2 Typical Areas of Conflict

Marketing and product management frequently find themselves in a state of conflict because of the strong interdependencies between the two roles. We highlight three common sources of conflicts and discuss ways to mitigate them.

Strategic Conflict: Brand Versus Product Investment
A typical marketing department is inclined to invest in overall brand development because this will lead to a more beneficial impact for their KPIs. On the other hand, investment in product marketing suits the product manager's KPIs better. The decision of how to distribute marketing investments among brand development and specific products is a strategic decision that should be made at the board level. Point this out to management when necessary.

Product Conflict: Roadmap and Requirements
The roadmap is a heavily contested document that virtually all departments want to influence. The marketing department is no different. Indeed, the number of interesting and potentially worthwhile ideas is typically much bigger than can be implemented. A product manager should say no to most these. At the same time, a product manager needs to be on the lookout for features that convey the 'allure of innovation.' Although these features will not drive sales, they demonstrate to your customer that you are an 'innovative player' or 'thought leader' that is worthy of their business.

Communication Conflict: Product Launch
The launch of a new product or product version requires careful orchestration of many different stakeholders within and outside of your organization. The marketing department is responsible for supplying all product promotion matters. However, asking them to drop everything and start working on a particular product when that product's next version is nearing technical completion will result in protest and conflict. It is best to make an organization-wide agreement on what is in a product promotion package and the expected time for delivering such a package. Note that this can take many different forms aside from the classical brochures and slide deck. For example:

- Press conferences accompanied by a press release.
- Media buys, both online and offline (Google ads, posters).
- Product logo.
- Media and texts for the website.
- Public relations deals with strategic partners.
- Inclusion in the organizational newsletter.
- Social media presence.

Approaches to Addressing Conflicts with Marketing
To avoid and mitigate conflicts with marketing, a software product manager can:

- Ask the Board to define trade-off criteria for brand versus product marketing,
- Involve Marketing early and frequently in product strategy discussions,
- Agree what a product promotion package entails and the time required for completion, and

- Have a regular jour fixe meeting with Marketing to exchange information and make decisions in a timely way.

6.3.3 SPM's Focus Areas for Orchestrating Marketing

There are some areas that an SPM needs to focus on when orchestrating marketing:

- Positioning of product in marketing plan,
- Plan execution,
- Product launch,
- Channel and partner management,
- Selective participation in marketing events, and
- Trading off brand marketing with product marketing.

Measuring Marketing's work and success have historically been difficult. The more marketing is focusing on the internet, the easier measurement becomes. Fine-grained marketing-specific KPIs that reflect customer behavior are collected to measure the success of the communication efforts. Examples are:

- Opens and clicks: the percentage of people that open communication messages and click on action links or buttons,
- Email submissions: the number of emails submitted each day/week/month,
- Requests for information: the number of requests for information (RFIs) or requests for proposals (RFPs) received,
- Number of new customer contacts,
- Number of new business opportunities,
- Online conversions: conversion of all or specific online channels, and
- Number of trade show leads: the number of people that can be contacted thanks to a trade show.

The marketing industry is claiming that they are turning themselves into a data-driven industry, and a software product management organization should embrace this trend.

6.4 Sales and Distribution

Companies establish a company-wide sales strategy within which the sales activities for individual products are planned and executed. The more a software company invests in sales, the higher the sales volume and profit margin usually are. The economic considerations described in Chap. 2 and the marketing leverage described in Sect. 6.3 explain this relationship. An important question is whether the main objective of the product strategy is growth or profitability. The former

maximizes the sales volume, the latter the number of licenses placed on the market. The answer depends on to the position of the product in its life cycle (see Sect. 4.4).

Marketing measures create the necessary *pull* in the market and stimulate demand. Sales activities provide the supplemental *push* so that products are purchased and contracts signed. Sales success determines the *top line*, i.e. the total corporate revenue and market growth. The actual split of responsibilities between Marketing and Sales may differ from company to company.

There are two approaches to sales: inside and outside sales. *Outside sales* involve interacting with customers face-to-face. Outside sales are common in enterprise sales, where the sales process tends to be more complicated due to the complexity of the problem space and customer decision making. *Inside sales* are made remotely with sales representatives interacting with customers via the phone or internet. As the Internet grows in importance as a channel to software customers and as more software is delivered as a service, the role of inside sales is growing.

Distribution means making the product available to the customer for use. Distribution can be under Sales' responsibility, or with a central fulfillment unit. Distribution can include the logistics needed to transport shrink-wrapped products to retail outlets, offer downloads on the internet, or deliver the software product on physical storage media to a customer after an acquisition. Manufacturing is usually not an issue for pure software products or services except for shrink-wrapped software. In the case of Software-as-a-Service, the provisioning of software in clouds, i.e. hosting, is the equivalent of manufacturing and supply of physical products.

The Sales and Distribution column in the SPM Framework lists the main tasks:

Sales planning addresses the development and negotiation of plans for all sales-related activities. Sales are planned for a pre-defined timeframe, often a year. A sales plan defines target values and incentives. The plans can be product-specific, or for groups of products. They need to be synchronized with corporate and product strategies and plans, and the Marketing plan.

Channel preparation means that the selected channels (see Sect. 3.6.3) are enabled in time to sell a new product, version, or release. It includes skills management and the provision of materials, website, and customer testimonials. Sometimes this responsibility is assigned to Marketing.

Customer relationship management (CRM) means the systematic management of a company's interactions with customers, clients, and sales prospects. CRM includes customer communication, knowledge management, and customer requirements engineering. CRM must not only focus on short-term sales success, but also on long-term customer relationships. Maintaining contact with existing customers is extremely important, since satisfied customers are a reliable source of information and future revenue, especially in the software industry. In comparison, winning new customers requires far more time and effort, but it is a process that is essential for further growth.

Operational sales mean the execution of the sales plan, tracking of the relevant measurements, and taking corrective actions when measurements deviate from the

plan. The execution includes offers and the negotiation of contracts, and the management of offers and contracts.

Operational distribution means ensuring smooth order and distribution processes, sufficient supply (in the case of physical distribution), meeting the terms of service level agreements (in the case of software-as-a-service), stable and easy online order and distribution, and smooth and correct billing and payment follow-up.

The product manager's orchestration responsibility includes the positioning of a product in the sales and incentives plan and tracking of plan execution. He has to be involved when product-specific commitments are given to customers. This involvement is particularly important when customer requirements are traded off between the short-term sales and long-term product perspective, and when Sales has to deviate from the standard terms and conditions, price levels, or price structure. Also, the product manager may decide to be selectively involved in pre-sales meetings with key customers.

Understanding the Sales organization and related processes is important to software product managers because, as part of their orchestration responsibilities, they should attempt to influence the motivation of the Sales organization for their product and ensure the salesforce is willing and able to sell their product.

6.4.1 Sales Motivation and Compensation

Compensation for sales professionals is different from that of other software organizational functions in that a significant proportion of their compensation is variable. Sales professionals are often compensated based on commission, which is a percentage of the sales revenue they generate. As part of the sales planning process, sales professionals can be given a *sales quota*, which is a monetary figure representing the minimum amount of revenue they are expected to generate from particular products or services. Setting quotas can be a powerful way to incent sales professionals to focus on specific products or markets.

Customers are sometimes given a discount on the purchase price of software products to encourage them to purchase it. Discounting is a powerful mechanism for sales professionals to convince customers to buy. Discounts, however, lower product-related revenue. We recommend strict governance rules for any pricing decisions (see Sect. 3.10). In any case, Product Management must monitor the discounting practices and balance the need to close sales with defined business objectives to manage this trade-off.

6.4.2 The Sales Organization

Sales organizations often report to a board level position responsible for sales or sales and marketing. Still, there is no single, widely accepted organizational structure for the Sales function within the company or for the Sales organization

itself. Sales may be given accountability based on a myriad of segmenting dimensions such as geography, industry, customer size, or a combination of them.

The Sales responsibility and activities of an account manager for existing customers are completely different from those required for new customer acquisition. The terms *farmer* and *hunter* can be used to describe these two different types of sales representatives. Depending on the size of the company and on the product, it can make sense to assign different persons to each of these two tasks or even to establish special Sales units for new customers.

A vertical sales structure, one that is organized by industries, has become increasingly popular and proven successful for both existing customer service and new customer acquisition. Today, practically every software product vendor whose target market is cross-industry, and most of the larger consulting and service companies have such a vertical sales structure if the company size and critical mass of the Sales organization permit this. The reason for the attractiveness of the vertical structure lies in market expectations: Many customers no longer seek information technology as such, but rather ICT-based solutions to improve their business processes. Therefore, vendors must be able to describe the advantages of ICT and its positive effects on the business processes of an industry or a single customer.

A customer does not expect the salesperson to be an expert in their industry. However, a customer expects that the salesperson knows the vendor's products and enough about the industry to explain how the offered products can be converted into financial benefits for the customer. The winning market player will be the one whose sales structures and processes can most convincingly communicate this type of conversion.

The disadvantage of a vertical structure is that it does not allow comprehensive, regional customer attention. In increasingly specialized ICT markets, the advantages of focusing on an industry clearly outweigh the disadvantages of not being able to provide regional attention. In some companies, the vertical sales organization even takes precedence over and has a more binding character than individual country or territorial organizations.

In a globalized world, it is no longer possible to give regional attention to key customers. It is increasingly becoming a major competitive advantage to have a Sales organization that operates on a global scale and is available to global customers. Many large ICT vendors have therefore implemented concepts such as *global account management* to provide a one-stop service to global clients. This approach is also attractive to conceal internal conflicts between individual Sales units from the customer.

6.4.3 The Sales Cycle

The process of identifying potential customers and selling them a software product is often described as a series of steps that are known as the sales cycle. The goal of Sales is to make the sales cycle as short as possible. Many definitions for the sales

Phase	Description
Prospect	Prospecting involves identifying potential customers
Contact	The contact phase represents communicating directly with prospects identified in the previous phase.
Identify needs	The sales professional gathers information from the prospect to identify pains and needs andassess if the vendor's offering can address these pains and needs.
Propose offer	The sales professional generates an offer. For complex solutions, e.g., enterprise B2B solutions, the sales professional relies on others, sometimes from the product team, to suggest the most attractive offer possible for the prospect.
Manage objectives	Once the offer is delivered to the prospect, objections to the offer often arise. For example, the prospect may feel the offer is too expensive.
Close	The sales professional generates a contract and gets binding commitment from the prospect to pay for the software product.

Fig. 6.1 Stages of the sales cycle

cycle are available from various sources, with many of them defining phases or stages like these (Fig. 6.1):

The process of a customer buying a product is often referred to as the buying cycle. It is important that the stages of the vendor's sales cycle are aligned with those of the customer's buying cycle, particularly for complex sales.

6.4.4 Typical Areas of Conflict: Sales

In most companies, the relationship between Software Product Management and Sales is conflict-laden. Typical areas of conflict include:

Getting Product-Related Commitments from Sales
While a product manager wants reliable commitments regarding the sales volumes of his product, Sales is typically only willing to commit numbers for larger product groups, but not for individual products. Consequently, the product manager often complains that Sales is not sufficiently focused on his product. Sales will claim that the product does not fulfill the current market requirements. Those conflicts can only be overcome if both parties are forced to commitments early in the development cycle of a product version. Such commitments can only be achieved if executive management enforces them.

Getting Product Feedback from Sales
Their contacts with prospects, clients, and competitors make sales professionals a valuable source of product-relevant information. The variable nature of their compensation and the inherent difficulty of their job prevent many sales teams to invest the time necessary for providing detailed feedback to the product team and the product manager. Product managers and their leadership must negotiate with

sales leadership sufficient time and rewards for engagement and underscore the critical and unique value of information coming from the sales organization.

Sales Incentives
Because of misaligned quotas and incentives, sales professionals may not be motivated to give some products the attention that the associated product managers expect. It is important for product managers to know and influence the product-related incentives to ensure their product is receiving the focus from Sales it needs to meet its business objectives.

Sales Price and Discounting
Pressure to close sales can incent salespeople to seek significant discounts and thereby lower a product's overall revenue and profitability. Product managers must monitor discounting policy and practice to ensure that discounting isn't compromising a product's overall business objectives. Differing discounting policies between products can result in some products' being heavily discounted to compensate for "discounting freezes" on others. Ideally, there are corporate pricing governance rules in place (see Sect. 3.10).

Short-Term Customer Requirements vs. Longer-Term Market Requirements
To secure sales, sales professionals may need to request features that a small number of customers, or even a single customer, requires. Product managers must consider the opportunity costs of such investments, primarily related to diminished investment in features that are broadly appealing. Features that are narrowly used can generate significant development and maintenance costs over the life cycle of the product and, consequently, increase complexity and erode profitability. The business impact of alternative investments in the product and the cost impact of implementing customer-specific features should be part of the product manager's arguments.

Impact of SPM Customer Engagement on Sales
It is important or even critical for SPMs to have direct contact with customers for a variety of reasons. These include the gathering of feedback and validating product-related plans. Customer contact should be coordinated with sales to reduce the risk of endangering ongoing sales negotiations or the appearance that the different organizations within the software vendor are not aligned.

Approaches to Addressing Conflict: Sales
Product managers should agree on a formal engagement model with Sales. That model should include regular, timely face-to-face meetings to discuss strategic product-related topics:

- Obtain the support of leadership to ensure that the Sales organization has the appropriate incentives in place to position and sell the product manager's product(s),

- Regularly review discounting policy and behavior to become aware of the impact of discounting on the product's revenue,
- Ensure that Sales understand the product strategy and market needs to set priorities when sales are blocked due to the requirements of a small subset of customers, and
- Actively keep the Sales organization apprised of customer engagement activities, including updating CRM or similar systems of record.

6.4.5 SPM's Focus Areas for Orchestrating Sales and Distribution

There are some areas that an SPM needs to focus on when orchestrating Sales and Distribution:

- Positioning of the product in the sales plan,
- Sales plan execution,
- Product-specific commitments to customers (features and/or measurements),
- Handling of customer requirements (short-term sales vs. longer-term product goals),
- Deviations from standard terms and conditions,
- Deviations from minimum price levels or price structure,
- Selective participation in pre-sales meetings,
- Skills of sales representatives,
- In bigger companies: sales representatives dedicated to product family vs. cross-product, and
- Alignment of sales measurements with SPM's responsibilities (product vs. product group focus)

Measuring Sales' work and success is easy. It can be measured in revenue, number of licenses or contracts, and market share, either new or total. The selection of measurements depends on the stage of the product's life cycle (see Sect. 4.4) and the company's strategic focus.

6.5 Service and Support

The service unit is responsible for the services that are offered and provided to customers. The offering includes both product-related and consulting services. The latter are not product-related, but customer-specific like custom software development or system integration, and therefore out of SPM scope.

Product-related *services* include education, installation, customization, operations, maintenance, technical support, and helpdesk covering technical and non-technical problems. Typically, these services are priced separately. Sometimes they are bundled with software product offerings.

The term *support* is often used for the services that help customers with product-related problems. In some organizations, support may be organizationally separated from other services. Also, there can be internal support for marketing and sales.

Product managers need to have a broad and deep understanding of customer pains to identify opportunities to address customer challenges with services. Services can be critical for customer adoption and retention, particularly for products that address complex problems.

The vendor as well as partners can provide services. Even companies that would prefer to relegate service delivery to partners can be compelled by customers to provide them as well as they expect a complete solution and don't want to establish an additional business relationship.

The Service and Support column in the SPM Framework lists the main tasks:

Service planning and preparation address the development and negotiation of plans for all product-related service activities during a given timeframe, often a year. The planning includes the definition of target values and incentives. The plans need to be synchronized with product strategies and plans, and the marketing plan. Preparation includes the development of a technical basis if required, the forecast of demand in collaboration with SPM, resource management, skills development, and development of marketing material in collaboration with Marketing.

Service provisioning means the execution of the service plan, tracking of the relevant measurements, and taking corrective actions when measurements deviate from the plan.

Technical support means the fulfillment of maintenance contracts. The typical support structure is:

- Level 1: Helpdesk,
- Level 2: Technical Maintenance, and
- Level 3: Change Team (typically in or with Development).

A call center on level 1 answers telephone calls or receives inputs from users. The call center staff does not need to have profound technical product knowledge but should be able to answer basic questions and differentiate between product failures and user errors. On level 2 dedicated product specialists deal with failure analysis and debugging. Level 3 usually involves the product developers, also known as the *change team*, and handles particularly difficult problems and fixing.

As a rule, 80–90% of all problems arising at the levels 1 and 2 should be dealt with on the respective level. Only 1–4% of all problems reported on level 1 should be permitted to reach level 3. Significant deviations from this rule of thumb, i.e. a higher percentage at level 3, require thorough analysis. Such deviations constitute a cost and resource problem as the change team at level 3 constitutes a rare resource whose other job may be coding new function. The deviations may be caused by a product quality problem or by level 1 and 2 staff members being poorly qualified, a problem that can be addressed by training.

The inputs obtained in the customer calls need to be categorized in bug reports or defects, feature requests, and non-technical problems. These customer inputs should be documented in a customer issue database.

Marketing support means providing help to Marketing. It may include the production and distribution of marketing material, organization and execution of marketing events (e.g. conferences and user groups), and documentation and tracking of marketing activities and their results.

Sales support means providing help to sales representatives and channels. It may include sales-related inbound and outbound communication with customers, typically through a call center. Also, it may concern the organization and execution of sales events (e.g. customer visits and meetings), documentation and tracking of sales activities and their results, and the provision of marketing material on request.

The product manager's orchestration responsibility includes the management of product-related services as part of the product offering, tracking of service execution, resource management, and skills development.

Product-related services include:

- Installation support,
- Customization,
- Operations (for SaaS offerings),
- Product training for various target groups,
- Helpdesk, and
- Technical support and maintenance.

Service and support produce product-related *documentation*. Such documentation is required both internally and externally. Internal documentation includes documents like specifications and technical manuals. Development usually develops this documentation, and it is not intended for customers. External documentation refers to all documentation intended for use by people outside of the vendor company like end users, ecosystem players, or service partners. For end users' guidance, printed and online manuals, help functions, or step-by-step instructions need to be specifically developed. This work requires collaboration between UX design, software development, technical support, and marketing. This work needs to be orchestrated by the product manager.

Another important element of product-related services is *product-specific training*. Training is offered to the company's sales and marketing, technical support, maintenance staff and, if relevant, to the customers. In larger companies, the provision of training requires a multiplier approach. Developers and product managers conduct initial training for the training staff (*train the trainers*) that subsequently conduct the training. Such elaborate training programs must be planned far in advance. The responsible product manager must ensure that the relevant organizational units allow for the training and provide in time the required workforce and budget.

Support services interact with customers after the sale, often at particularly critical times, for example, when an issue is preventing the software from

performing important functions. Service organizations also get deep insight into how customers are using the software product and, thus, which parts of the product cause customer dissatisfaction. It is for these reasons that it is important that SPMs regularly engage with support. Product managers should seek to understand support requirements and challenges and prioritize features that improve supportability based on sound business criteria. Although supportability enhancements don't generate the same customer excitement as features aimed at end users, such investments can improve customer satisfaction. As a result, support costs become lower, and the business performance of the software organization improves.

Also, service and support professionals need to understand the product's value proposition and roadmap and should know to whom to refer a customer if they discover product-related opportunities.

Although services can play an important role in providing good customer value, the *definition and management of services* require knowledge and skills that differ from those required to manage software products. Product managers should play a key role in shaping the product-related service strategy (see Sect. 3.7), identifying services opportunities, and defining service-related requirements for their product. Nonetheless, they should engage other professionals with the appropriate level of experience to do the detailed services definition, planning, and delivery.

Since Service organizations focus on various aspects of delivery and maintenance, they may report to the sales organization. Or, Services may be set up as a peer organization to sales and marketing or report to another organization entirely. It is common for Services organizations to have a planning cycle and business model that is independent of the product organization (see Sect. 2.4).

6.5.1 Typical Areas of Conflict: Service

Service and product management sometimes find themselves in a state of conflict. We highlight the following sources of conflicts and discuss ways to mitigate them.

Getting Feedback About the Product from Services
Because they often help customers deploy and use software products, services personnel are a rich source of feedback about the product. Unfortunately, people in the Services organization are often incentivized to spend as much time as possible working on customer projects. This work focus leaves them and their leadership with little incentive to invest time in providing feedback or otherwise engaging with SPMs. SPMs should seek to develop relationships with members of the services organization and should lobby for a reasonable amount of time per year to provide feedback to the product team.

Sharing the Product Roadmap/Positioning with Services
As previously noted, Services can spend a considerable amount of time with customers before and after the sale. They may, therefore, be able to position new features or offerings with customers. For this reason, product managers should

invest time in sharing the roadmap and associated positioning with the Services organization.

Prioritizing Features for Markets vs. Individual Customers
The Services organization's involvement in customer projects may incentivize them to advocate for product features that have low appeal to other customers. SPMs should be prepared to explain investment decisions to Services personnel and underscore the necessity of meeting the needs of multiple stakeholders like customers.

Product Enablement for Services Personnel
Product managers should ensure that Services personnel receive adequate information and training about the products, especially in conjunction with new releases. Inadequate enablement may result in low-quality work on their part on customer projects or the perception that others, e.g. third-party services providers, have superior product knowledge.

Provisioning and Hosting
With Software-as-a-Service offerings, the software organization is responsible for hosting services. A lot of companies work with external hosting providers. An in-house hosting approach offers opportunities for improving time-to-market with a DevOps approach (see Sect. 6.2.1), however. The key success factor for implementing DevOps is the definition of fast and flexible collaboration between Development and Operations.

Product Team Support for Services Organizations
The Services organization may need support from the product team for solution design and bug resolution. Product managers should attempt to provide the Services organization the help it needs while minimizing the impact on product development.

Ensuring Customer Satisfaction with Support Issue Resolution
Support professionals and organizations are often measured on how quickly they resolve customer issues. The KPI can incentivize them to mark support incidents as closed before the customer is satisfied with the resolution. Frequent engagement with the Support organization and the definition of guidelines or practices indicating when SPM should be made aware of important issues can help SPMs to manage this risk.

Balancing Investment in Supportability Features with Other Priorities
A product should be easy to support. The required investments must be balanced, however, with investments that ensure appropriate market demand. All too often, investments in supportability receive lower priority than features that customers request. Unbalanced decisions result in low business performance due to support

costs. SPMs should seek to assess investments in supportability based on business criteria.

Ensuring Support Professionals Have Adequate Knowledge of the Product
Support professionals must be trained on how the product functions from the perspectives of the end-user and backend. For customer satisfaction, it is in the SPM's best interest to ensure that the Support organization understands not only the technical aspects of the product but also the business goals of the product. Such understanding gives them context that can be important in understanding the nature and magnitude of customers' support issues. The product team may need to supply product documentation aimed at Support professionals in additional to documentation intended for customers.

Timely Communication of Support Issues that Impact Customer Satisfaction
SPMs should work with the Support organization to define guidelines for when SPM should be made aware of issues that may affect customer satisfaction or business performance. These issues must be routinely brought to the attention of SPMs that make the appropriate business decisions. Also, since SPMs often engage with a customer, it is important that they not be caught unaware of critical issues.

6.5.2 Approaches to Address Conflict: Service and Support

To avoid and mitigate conflicts with Service and Support, a software product manager should:

- Create formal plans to ensure Service personnel understands both the business/functional and technical aspects of the products, including positioning and appropriate hand-offs for customer queries regarding product roadmap and strategy and the value of the overall portfolio,
- Define reasonable practices to ensure that development capacity is not lowered thus impacting business objectives by direct queries from Service to Development,
- Make themselves available to help Service personnel to navigate the Development organization to ensure timely resolution of product-related issues encountered by the Service organization,
- Explain Support professionals the important role they play in customer satisfaction, so they can better assess tradeoffs between this and support goals such as closing open issues as quickly as possible,
- Play an active role in upskilling Support with each new release, including the business motivation,
- Invest time in working with or "shadowing" Support to understand supportability requirements and Support's daily challenges better,
- Elicit supportability requirements from the Support organization and involve the Support organization in release planning, and

- Collaborate with Support to define reasonably clear criteria that trigger Support to contact SPM regarding issues that are deemed relevant.

6.5.3 SPM's Focus Areas for Orchestrating Service and Support

There are some areas that an SPM needs to focus on when orchestrating Service and Support:

- Consider and manage product-related services and documentation as part of the offering,
- Service execution,
- Skills of Service specialists,
- Frequent analysis of incoming service calls (often good indicators for problems with quality, usability, and functional coverage), and
- Resource management (avoid bottlenecks that impact product development, sales, and customer satisfaction).

Service units are typically measured on revenue and customer satisfaction. They usually follow a service business model (see Sect. 2.4).

6.6 Orchestration Skills

6.6.1 Virtual Teams and the Matrix Mindset

Product managers are challenged to achieve "influence without authority, accountability without control" [Hall13]. This challenge is the same that managers in matrix organizations face.

A certain mindset and leadership skills are required to succeed in such a setup. Hall compares two very different mindsets regarding matrix work: a successful matrix manager "relishes the flexibility, autonomy, and breadth that the matrix gives them." This statement describes the approach that helps software product managers succeed in orchestrating other functions. Someone believing that things can only get done in a traditional organizational structure, "where managers thought they had all the answers and cascaded clarity, authority, and responsibility" will find it difficult to succeed in a matrix organization. This mindset is the one of the matrix victim, which is not helpful for orchestration tasks.

Hall summarizes key beliefs that characterize these two different mindsets (Fig. 6.2):

Matrix Victim	Successful Matrix Manager
My goals are not clear.	Here are the commitments I have chosen.
I do not have a job description,and I am not clear what I should be doing.	This is what needs to be done.
I do not have the authority to get things done.	Who do I need to influence to get this done?
I cannot be accountable for things I do not control.	Where can I get the resources to meet my commitments?
My manager doesn't empower me.	What have I done to earn the right to take on more responsibility?

Fig. 6.2 Beliefs about matrix organizations [Hall13]

6.6.2 Sources of Power

Because the various company functions rarely report to the SPM organization, software product managers must find creative ways to influence to achieve product vision, goals, and objectives. Understanding what power is, where it comes from, and how to increase it can help software product managers increase their influence.

Social communication studies have theorized that leadership and power are closely linked. They have further suggested that some forms of power affect one's leadership and success. That idea is often used in organizational communication and throughout the workforce.

The social psychologists French and Raven released a study in 1959 [FrenRav59], in which they define the terms *power*, the ability to influence others, and identified the *bases of power* described in Fig. 6.3.

Product managers typically exert the most influence using referent and expert power. Depending on the organizational setup, their legitimate power is limited, and they have little or no reward or coercive power over other organizational functions.

6.6.3 Managing Conflict

Product managers are expected to orchestrate across the organizational boundaries of an organization, without the power to direct other business functions. Consequently, conflict with other business functions will occur that prevents product managers from reaching their goals. Nevertheless, conflict is also beneficial for organizations. Disagreement necessitates two parties to negotiate to find a single, mutually satisfactory solution [ZarRub00], a solution that is often of superior value than if it would not have been negotiated [RaiRicMe07]. This chapter introduces selected approaches to conflict resolution, the human-process and techno-structural approaches [Rahim15], that include the possibility of intervention by a third party after escalation.

Basis	Characterization
Reward power	Reward power is based on a person being able to control the likelihood that another person will be rewarded. Close alignment of SPM with executive leadership may offer SPM some reward power.
Coercive power	Coercive power uses the threat of force to gain compliance from another with physical, social, emotional, political, or economic means. Typically, SPMs have very little coercive power over other functions as they have no direct management authority over them.
Legitimate power	Legitimate power comes from an elected, selected, or appointed position of authority and may be underpinned by social norms. An employee's direct manager often can exert legitimate power because the organization has authorized that manager to influence the people that report to him or her. SPMs typically have some legitimate power granted to them by the organization. Some development life cycle models, like Scrum, empower the product owner, who may report to the product manager, by giving the product owner ownership of the product backlog.
Referent power	Referent power is based on one person's strong identification with another. A person identifying with another person is likely to adopt attitudes and beliefs similar to that other person. Referent power is often the primary source of power for software product managers. Product managers can increase their referent power by defining and executing a compelling product strategy and investing in trustful relationships with other organizational functions.
Expert power	Expert power is based on one person's perception of another's knowledge in a given area. For example, a person's thinking and behavior regarding a legal case might easily be influenced by advice from a lawyer. SPMs often have expert power based on their knowledge of stakeholders, the business aspects of the software product, and the domain addressed by the software product.

Fig. 6.3 Bases of Power [FrenRav59]

The *human-process approach* "attempts to improve organizational effectiveness by changing members' attitudes and behavior regarding a conflict." The approach suggests achieving this goal by educating the concerned stakeholders on the five styles of handling interpersonal conflict. Figure 6.4 gives an overview, and Fig. 5.9 briefly summarizes each style, differentiated on two dimensions: concern for the self (how much a person attempts to satisfy his personal interests) and concern for the others (how much a person wants to satisfy other parties' interests) (Fig. 6.5).

The techno-structural approach attempts to "improve organizational effectiveness by changing the organization's structural design characteristics" or managing the amount of conflict by introducing organizational changes. A variety of organizational change techniques exists. Weiss and Hughes introduce a comprehensive set of best practices for managing disagreements at the point of conflict [WeiHug05]. These practices enable employees in resolving conflicts themselves. Weiss and Hughes also suggest approaches for managing conflict upon escalation up the management chain. When necessary, employees escalate a conflict to a superior, who decides on the employees' behalf. Weiss and Hughes recommend implementing three tactics for each type of conflict resolution. These tactics transform conflict from a major liability into a significant asset. We summarize

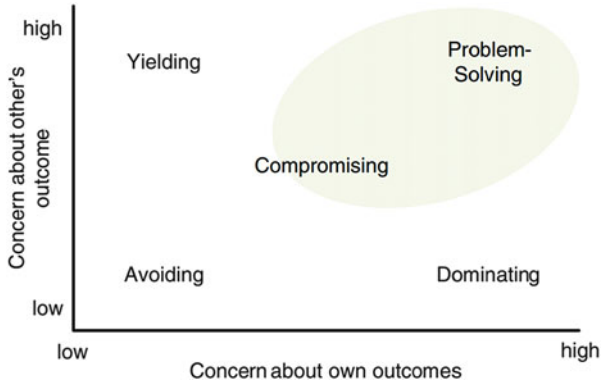

Fig. 6.4 Styles of handling interpersonal conflict, [FricGru08] based on [Rahim15]. *Grey-shaded* desired, value-creating behaviors

Yielding low concern for self, high concern for others	A party sacrifices his personal interests to satisfy the interests of the other party. The former reacts to a perceived hostile act of the latter with low hostility or even positive friendliness. During a negotiation, the former will attempt to play down the differences and emphasize commonalities between the two parties.
Dominating high concern for self, low concern for others	A competitive or dominating individual will do anything to achieve their personal objectives. Consequently, he or she ignores the needs and expectations of the other party. A dominating person tries to impose his will by sheer force, e.g. on subordinates, and commands their obedience.
Avoiding low concern for self, low concern for others	A party avoids negotiation by "I see no evil, hear no evil, and speak no evil." The party attempts to postpone an issue, withdraw from the situation, and refuse to acknowledge the conflict. When the conflict is avoided, the interests of neither party are satisfied.
Problem-Solving high concern for the self, high concern for others	Leads to full collaboration between parties. All try to be open, exchange information, and examine differences to reach a mutually acceptable solution. This style has two distinctive elements: confrontation and problem solving: "confrontation involves open and direct communication which should make way for problem-solving. As a result, it may lead to creative solutions to problems."
Compromising medium concern for self, medium concern for others	Compromising is characterized as give-and-take or sharing and often used to conclude negotiations rapidly. Both parties give up something to make a mutually acceptable decision. They might split the difference, exchange concessions, or search for middle ground. This style is in the middle of all other styles.

Fig. 6.5 Styles of handling interpersonal conflict, based on [Rahim15]

the tactics in Figs. 6.6 and 6.7. Be aware that the applicability of escalation depends on the corporate culture of an organization.

Devise and implement a common method for resolving conflict
A company-wide process for resolving disagreements prevents useless debate about who is right or wrong and haggling over small concessions. A well-designed conflict resolution method will reduce transaction costs and foster an environment in which innovative outcomes emerge from discussions. Unfortunately, no conflict resolution method is universally applicable. A conflict resolution method must offer a clear, step-by-step process and integrate the process in existing business activitiesto be effective. Exceptionally used processes will be unsuccessful.
Provide people with criteria for making trade-offs
From time to time, two parties need to make zero-sum trade-offs between competing priorities – situations in which it is unclear which decision is best. The criteria for making such choices must be defined for the whole organization. E.g., salespeople that know that 5 points on a customer satisfaction scale are more important than 10 points of market share are capable of making informed decisions. The criteria will foster productive discussion of common objectives.
Use the escalation of conflict as an opportunity for coaching
Senior management should view each escalation of conflict as an opportunity to teach employees about how to resolve conflicts effectively. Management should push their respective employees to consider "the needs of the other party, alternatives that might best address the collective needs of the other party, and the standards to be applied in assessing the trade-offs between alternatives." While this approach requires time from senior managers initially, it reduces the time senior managers need to spend on resolving escalated conflicts in the long run.

Fig. 6.6 Tactics for managing disagreement at the point of conflict

Establish and enforce a requirement of joint escalation
In many conflicts, the parties try to get support from their direct leadership. Prevent a vicious circle by enforcing people to present disagreements to their management together. Now, the decision maker has a balanced view of the perspectives on the conflict, its causes, and possible solutions. Moreover, the number of problems that are escalated decrease.
Ensure that managers resolve escalated conflicts directly with their counterparts
An unresolved dispute tends to travel up the management chain until a senior manager with the appropriate organizational influence makes a unilateral decision. This dynamic breeds organization-wide resentment in the form of "we'll win next time, " making friendly conflict resolution increasingly difficult. Moreover, unilateral decisions lead to inefficiency, ill feelings, and bad decision-making. Instead of propagated escalation, managers should formally commit to dealing with escalated conflicts directly with their management counterparts in other departments.
Make the process for escalated conflict resolution transparent
Having resolved a conflict, managers at most companies announce the decision and move on. This behavior prevents employees from learning how to resolve similar issues in the future. Management should explain instead how the specific aspects were weighed and how the decision was reached. No tevery single detail of the process needs to be shared, but the relevant trade-offs honestly discussed and people enabled for resolving future conflicts.

Fig. 6.7 Tactics for managing conflict upon escalation

6.6.4 Negotiation Skills

Negotiation is a discussion for reaching a shared agreement among multiple parties. As a leader in the product organization responsible for aligning the requirements and efforts of multiple stakeholders, negotiation is a critical skill for most software product managers. Product managers negotiate regarding the following topics, among others:

- Release scope and timing with executive management, Development, and others.
- Budget for functions like Marketing and Sales.
- Contracts with third parties like software suppliers or service providers.
- Product pricing and discounting with individual customers.

The Harvard Negotiation Project developed the concept of *principled negotiation* as described in their seminal work on negotiation, "Getting to Yes" [FiUrPa11] that was originally published in 1981 and elaborated into a systematic approach to negotiation [RaiRicMe07]. The concept addressed what they considered non-productive negotiation approaches and tactics. Principled negotiation encourages negotiators to bargain over interests rather than positions. It defines four principles:

- Separate the people from the problem: by nature, people often conflate the relationship with the substance of the conflict. Fears and perceptions can be turned into better understanding by discussing each other's emotions.
- Focus on interests, not positions: a position is what a party in a negotiation is willing or unwilling to accept. Interests represent the party's wants or needs.
- Invent options for mutual gain: creatively think of options that address negotiation parties' interests.
- Insist on using objective criteria: these criteria provide a means of resolving differences in interests outside the context of the negotiating parties' wills.

Principled Negotiation also defines the concept of a *Best Alternative to a Negotiated Agreement (BATNA)*. The BATNA is the most advantageous alternative course of action a party can take if negotiations fail and an agreement cannot be reached. A party in a negotiation should typically not accept an outcome that is less desirable than their BATNA. It is important to not only consider your BATNA when negotiating, but also that of the other parties.

SPM Today and Tomorrow

7

Companies are usually founded to participate in the economy as organizations that exist in the long term. In spite of all modernistic focus on the short term, it must be a key objective of any executive management to make the company's success sustainable. Based on this concept of sustainability, this book emphasizes the importance of state-of-the-art software product management for the success of companies that have software(-intensive) products in their portfolio.

In this last chapter, we want to look into the future of SPM (Sect. 7.1), derive conclusions from analyzing the state of practice (Sect. 7.2), and describe how SPM can be applied in different business scenarios (Sect. 7.3). Finally, we present ISPMA, the International Software Product Management Association, and how you can get value from ISPMA (Sect. 7.4) as a software product manager or a company.

7.1 The Future of SPM

Since software is not only becoming more and more pervasive in the form of standard software products, but also as embedded software in other industries' products and services, the value of software as a critical asset is increasingly recognized. This asset plays a major role in the sustainable economic success of the respective companies and therefore needs to be managed in a comprehensive business-driven way, which is exactly what Software Product Management (SPM) is about. With trends like Cloud, Internet of Things, Industry 4.0, fifth-generation telecommunication networks (5G), smart systems like self-driving cars etc., software is becoming the number 1 value driver in more and more industries. Harvard professor Michael Porter analyzed this development in recent articles [PortHepp15, PortHepp14]. We are seeing more and more tightly integrated systems with software, hardware, telecommunications (ICT), etc. that redefine the relationship between human beings and technology.

H.-B. Kittlaus, S.A. Fricker, *Software Product Management*,
DOI 10.1007/978-3-642-55140-6_7

With the growing importance of software, we are convinced that the establishment of and focus on software product management will enable the companies in all these industries to cope with future business and technological challenges in a better way. And challenges will abound. Most companies in non-software industries are facing severe cultural and skill issues that make it difficult for them to embrace the opportunities that software can open up for their businesses. Serious change management is required top-down, i.e. from executive management, but also bottom-up. Here software product managers can help as change managers who keep the organization on track towards "software thinking".

Corporate IT organizations have often considered themselves as project-driven service organizations. There is increasing awareness that a corporate IT organization's portfolio of software applications constitute sustainable assets not only for IT, but for the corporation as a whole. So a pure project view is not sufficient. They require a life cycle view that can be institutionalized through the establishment of SPM. In total, these developments broaden the applicability of the concept of SPM across industries significantly.

In Asia and other regions in the world where the software industry has been primarily focused on providing offshore outsourcing services, there is increasing interest in establishing a software product business. Reasons are that offshore outsourcing does not grow as much anymore as it used to, and that in a lot of these regions there is growing demand for localized standard software products and internet services, e.g. in India. Companies who want to address this demand need to establish product organizations including the role of software product manager, and need to change their business model and culture (see Sect. 2.4).

The combination of all the new technologies produces unprecedented amounts of data about customers and things (as in "Internet of Things"). So access to this data, aggregation and interpretation become key capabilities within companies. This creates opportunities for "data products", i.e. offerings by vendors where the main deliverable is data. Such products already exist for certain markets like stock markets or pharmaceutical markets. It is an interesting area for research and consulting how the concept of software product management can be adapted to become applicable to the management of these data products.

A software product management survey conducted by Andrey Maglyas and Samuel Fricker in 2014 produced some results regarding the future of software product management ([MaglFri14], quotes from participants in quotation marks):

All respondents agreed that SPM will play an important role in the future. With advances in technologies and tough competition in the market, empowered, systematic, and consistent SPM is important for software organizations to excel. Overall, based on the answers from the respondents three directions on the future of SPM can be identified:

Increased Awareness and Importance

In many companies SPM is still immature today [MagNiSmo12]. Companies try different approaches to adopt software product management practices in order to deal with constantly changing markets and technology trends, but the definition of

role and responsibilities of product managers is still far from being mature, understood and recognized [MagNiSmo13].

> There will be a better understanding of how important the product manager's contribution is to the success of software-based offerings (especially given that the technical feasibility is becoming less of an issue). Therefore, product managers will be increasingly recognized.

Certification, Standardization, and Education
As one of the respondents said

> It [software product management] will be more formalized, since the right product management is probably the most efficient investment you can make in a product, regardless of its place in the life cycle.

The product manager can be seen as an example of a middle manager who acts as a "linking pin" connecting different parts of the organization [FloyWool94]. In this position, product managers act as interpreters and implementers of the decisions, but also mediate between strategic and operational levels [MagNiSmo13]. This requires a deep understanding of the role and responsibilities of SPM in general and the ability to map this to a company-specific organizational structure and work practices of functional units like marketing, sales, development, and support. As long as product managers learn everything by doing, they frequently miss best practices already known in other companies. Therefore, education in product management will be the next milestone in accepting and spreading the discipline from self-learning to industry-wide practices. Serious certification will certainly help in this process both to encourage personal development and to help companies to identify qualified software product managers.

More Authority
The term "authority" can be interpreted in two different ways:

- Hierarchical authority in a management hierarchy enabling SPMs to implement any decisions they make with their own team reporting to them.
- Personal authority based on personality, experience, recognition and respect.

Product managers usually have no direct subordinates, and this differentiates them from other middle managers. This means that their hierarchical authority is limited and their role may be restricted to the role of advisor only [MagNiSmo13] or "cross functional leadership with no authority." Many of the respondents expect having more authority in the future. If they mean hierarchical authority we interpret this as wishful thinking only and do not see any signals for companies giving more hierarchical authority to product managers in the near future, exceptions granted.
If they mean personal authority we agree with this assessment. The more executive management values the SPM role and communicates the corresponding positioning accordingly, the better are the prerequisites for a well-educated and experienced SPM to achieve a higher level of personal authority. This in turn makes it easier for

an SPM to convince and influence customers, executives, and colleagues in the organization.

For the IT industry, prices have been going down so much that processor and storage capacity and communication bandwidth have already become commodities. Cloud computing is firmly established and shows continuous significant growth rates. In a lot of areas, capacity considerations are no longer the limiting factors that they used to be. This opens the door to a new phase of innovation with software as the key component. After a phase of about 15 years in which innovation was very much driven by the consumer business, innovation areas like Internet of Things, Industry 4.0, Big Data e.a. are driven by the enterprise business side again.

It is increasingly difficult for corporate IT organizations to stay on top of all these changes and innovations. Technology-driven start-ups come up with specialized new business processes and quickly take away market share, e.g. fin-techs in the financial services industry. "Digital Natives", the generation that has been used to PC games and the internet before they could even talk, are entering the workforce. They expect the same IT capabilities at their workplace that they are used to at home. They will not join a company that does not support their work style and life style adequately. Plus a company cannot benefit from their abilities to the full extent if it does not provide the appropriate environment. So the move to "Bring your own device" that a lot of companies have implemented can be seen as a response to this challenge, but it will not be sufficient.

The technological changes will continue to lead to innovative new business models some of which will have significant disruptive power. The role of software product manager will become even more important for continuously bringing together technology and business anew, not only in software vendor companies and corporate IT organizations, but across all industries.

These developments provide hugh opportunities for research. We can only list some topics as examples:

- Analytics for software product management.
- Correlation between a product manager's authority and product success.
- Correlation between software pricing approach and product success.
- Correlation between ecosystem strategy and product success.
- Software category building in different product scenarios and business scenarios.

7.2 The State of Practice

The role of the software product manager is firmly established in North America. One can hardly find any software vendor company that does not employ such a role. In Europe, the majority of software vendor companies have that role as well. Even in Asia/Pacific and other parts of the world, where the role is not commonly established yet, we see examples of strong companies like Samsung or some Chinese companies that have adopted the role. Also, there is a growing trend

towards adoption in corporate IT organizations and companies in non-software industries that produce software-intensive products and services.

While the term "software product manager" is most often used for the role, there are many other terms used as well (see Sect. 2.6). Also, we observe differences in the way the role is defined and lived (see Sect. 2.6), often a consequence of the specific business environment (see Sect. 7.3).

ISPMA Fellow Members Andrey Maglyas and Samuel Fricker conducted a software product management survey in 2014 [MaglFri14]. They collected the following quotes from participants in quotation marks):

Skills and Education

The need for SPM knowledge, but a lack of specialized education was constantly claimed by the respondents, e.g. "Even small software companies can now build business applications just like Oracle, SAP, etc. but typically the smaller companies do not have trained SPM professionals. Therefore the training becomes more important."

The respondents considered global competition, new markets, lowered entry barriers, and increased importance for sustainable strategy as the main drivers for the need of SPM education to acquire and train new skill sets to manage software products. Another lacking skill identified by the respondents is orchestration. Lacking skills product managers are learning by doing which can have a negative effect on the product.

To decrease the number of failures and bad decisions made by immature product management and to make the hiring process easier and more convenient, the respondents have proposed standardization and professionalization as a solution. The standardization should help to improve the "lack of consistency or consensus about product management roles and responsibilities across companies." Product managers have difficulties in finding comprehensive information work guidelines online. Standardization and professionalization may also address that need for knowledge.

As SPM is a multidisciplinary field, companies struggle with defining the skill requirements when appointing new product managers.

> Companies don't know how to hire product managers. They tend to focus on just one facet of a multifaceted job. For example, some will look for domain knowledge while others look for computer science training. I think an educational path would provide some degree of clarity.

Software Product Management Challenges

Lack of education and standardization in SPM are not the only challenges today. Although the respondents reported unclear definition of the role as one of the main challenges, we tend to consider this lack of clarity as a result of other challenges like too many responsibilities and little authority. With too many responsibilities, a product manager can easily keep herself busy with purely tactical activities like endless customer meetings and lose the focus on strategic activities like

roadmapping and product strategy. Another risk of being a "multipurpose person" is receiving more and more requests to fill many tasks ranging from development to user experience design to marketing, sales and support. However, while being in charge of many activities critical to the product success, product managers rarely have authority in practice. They are "often not sufficiently empowered by management and cannot make market-facing decisions or resist development's technically motivated agenda." Such lack of authority is a characteristic that distinguishes technically oriented product managers from more business-oriented product managers or senior product managers [MagNiSmo13].

The other reported challenges were related to general challenges of development and management of software products: customer understanding, rapidly changing environment, prioritization, coordination, and resource management. Regarding resource management, product managers rarely have their own resources and must make a request to higher management every time he or she needs extra resources for the product [MagNiSmo13]. So product managers need to identify resource gaps and predict a shortage of resources in order to request them from higher management in time.

Software Product Management Activities

To bring some insight to SPM in practice, the respondents were asked to name the most important SPM activities to manage a product properly. They mentioned: market analysis, requirements management, communication with stakeholders, customer analysis, roadmapping, orchestration, product life cycle management, product strategy, prioritization, vision, product planning, and product analysis.

All of these 12 activities are explicitly or implicitly contained in the ISPMA SPM Framework as shown in Fig. 7.1.

Most companies focus on a subset of activities prescribed by the framework, primarily in the area of core SPM activities. However, in the area of Product Strategy a number of activities listed in the ISPMA SPM Framework are not covered by a lot of product managers. This is a serious gap that companies need to address if they want to become more successful.

The fact that the survey results show Customer Analysis as a key activity can be interpreted in two ways. One is that knowing the market and what customers are doing with the product is highly important for product managers. The other one is that in some companies SPM tasks are mixed with marketing tasks.

> Having time to actually be a product manager—roles in every company are so different but overall, product managers seem to be the all-in-one job role, putting down fires, solving crisis, running after budgets and resources, which leaves little to no room to actually know what your roadmap should look like.

Therefore, prioritization is not only relevant with regard to product requirements, but also for the self-organization of the SPM.

In general, there is no single magic recipe for the definitions of responsibilities and the organization of software product management. These questions have to be

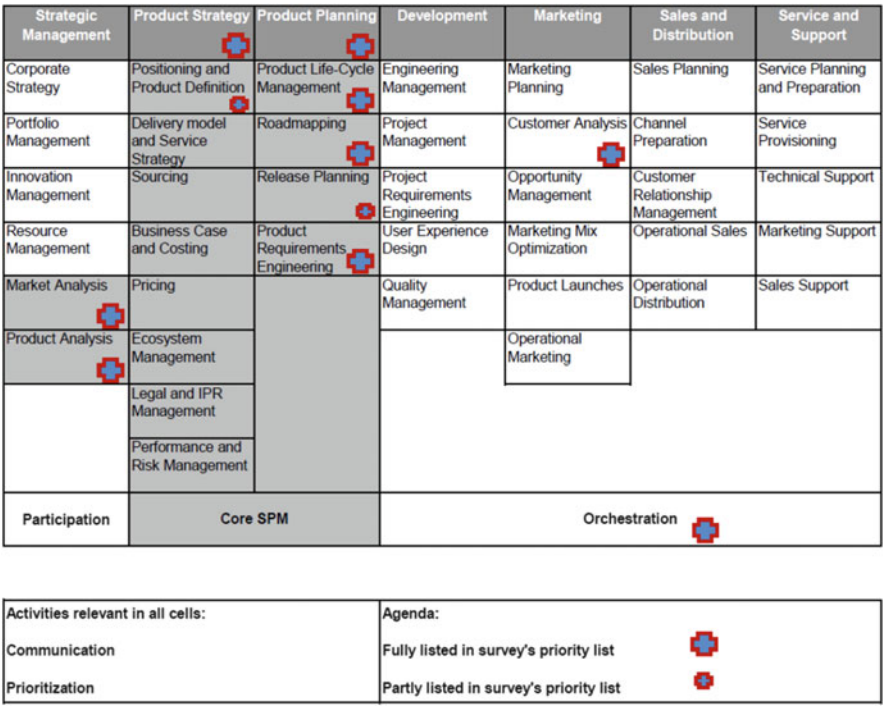

Fig. 7.1 Mapping of survey results into ISPMA SPM Framework

answered dependent on the objectives of the company, the products to be managed, the existing organizational structure and the company's culture. We argue that Software Product Management has the responsibility for the sustainable success of a product in the market. Success depends on allowing the software product managers to focus on the important items and not be overwhelmed by the urgent day-to-day necessities.

7.3 SPM in Different Business Scenarios

The concept of software product management described in this book and in ISPMA's syllabi is business-context-agnostic and therefore applies to a wide range of industries. However, there are some specific considerations in certain business contexts as described in the previous two sections of this chapter. These business scenarios are considered:

- Standard software products.
- Software in software-intensive technical services (e.g. internet platforms or SaaS).

- Software in software-intensive systems (embedded software).
- Software in professional (human) services (embedded software).
- Software managed by Corporate IT organizations (for one or multiple internal customers).

7.3.1 Standard Software Products

In this scenario the full contents of the SPM Framework is applicable.

7.3.2 Software in Software-Intensive Technical Services

The amount of software used in services like Software-as-a-Service (SaaS) or for internet platforms like community, communication, shopping, or contents platforms is increasing at a high pace. In all these instances the software contributes a significant part to the value proposition of the entire service. Therefore it ought to be managed from a software product management perspective. Actually, we consider SaaS offerings as software products (see Sect. 2.4). Often, the product manager of the technical service is also responsible for the software product management.

In these environments, agile methodologies are widely used. The relationship between SPM and the Product Owner is described in Sect. 6.2.1.

Most strategy aspects, in particular business aspects, are managed on the service level. Here the full contents of the SPM Framework is applicable. If standard software products from other vendors are used the service provider needs license contracts with licensors of the software which explicitly allow this kind of use. Customers of such a service do not need license contracts, but only service contracts with the service provider.

7.3.3 Software in Software-Intensive Systems (Embedded Software)

The amount of software embedded in hardware components and systems is increasing at a high pace (see Sect. 2.2). Software is turning into the number 1 value driver in more and more industries. Traditionally, hardware manufacturers had a life cycle view on their products that differed significantly from software vendors. With hardware, when the development of a version of a product is finished, the product goes into production. For an extended period of time, there is no further development, but only after sales services that take care of defects in individual product instances, e.g. individual cars. Then after quite some time, a new development project is started for the next version of the hardware product, and the cycle starts anew. If the hardware product includes software components, hardware manufacturers tended to treat this software in the same way as the hardware.

These days, this approach does not work anymore for more and more hardware products, because there is an increasing necessity for frequent software changes. So the life cycles of the software components become more similar to those of standard software products while the life cycles of the hardware components do not change so much. Actually, cost considerations motivate manufacturers to keep the hardware components as stable as possible. So from a product management perspective, hardware and software components need to be managed at two very different speeds (see also [LickKitt16]).

This difference means a significant challenge for a product manager who is responsible for the complete system of hardware and software components. The software components need to be managed from a software product management perspective. If there is a product management team it makes sense to establish dedicated software product managers.

Most strategy aspects, in particular business aspects, are managed on the product level. The software product manager will focus on positioning with regard to the other components of the product; on the scope of the software; on the business aspects directly related to the software part, e.g. business cases and costing; and on make or buy decisions.

7.3.4 Software in Professional (Human) Services (Embedded Software)

As we have pointed out in Sect. 2.4, the business models of a software product business and a professional service business are fundamentally different. In particular, there are limits to the profitability of a professional service business that do not exist for a software product business. That is why professional service providers in all industries are trying to find ways how to improve their profitability by replacing humans through standard software components in their professional service offerings. Sawhney has analyzed this approach in detail [Sawhney16]. He calls the software components products in order to point out that they need to be standardized and managed like software products, but they are not software products according to our definition since they are not sold as standalone products.

The software components need to be managed from a software product management perspective. Depending on the type of service, it may be too big a challenge for the business manager responsible for the service to assume the software product management tasks as well. We recommend a dedicated software product manager who works in tight cooperation with the service business manager.

Most strategy aspects, in particular business aspects, are managed on the service level. The software product manager will focus on positioning with regard to the other components of the service; on the scope of the software; on the business aspects directly related to the software part, e.g. business cases and costing; and on make or buy decisions.

7.3.5 Software Managed by Corporate IT Organizations

A growing number of corporate IT organizations in all industries is adopting the concept of software product management. The software components, in particular applications, in an enterprise architecture tend to have very long life cycles which require a strategic view and continuity in management. Both cannot be ensured in a pure project organization. Some corporate IT organizations have been transformed into profit centers and may have multiple customers inside and outside of the corporation for the same software components which makes their business model more similar to a software vendor.

In corporate IT organizations, the role of software product manager sometimes has different names, e.g. application manager or (application) service manager (see Sect. 2.6). Strategy aspects need to be managed in close cooperation with the companies, business units or departments in the corporation that are the customers of the software product. A most important aspect is the positioning of the software product in the enterprise architecture of the corporation over time. The relevance of the business aspects depends on how the business relationship between the corporate IT organization and the companies, business units or departments in the corporation is designed. If the corporate IT organization is run as a cost center its focus will be on business cases, budgets and costing, while the overall business responsibility is on the business side. The more the corporate IT organization is run as a business unit of its own, the more relevant the other business aspects become.

The relevant ecosystem is usually restricted to the technology side, i.e. software and other technology providers and software development partners. Since a corporate IT organization usually has the responsibility for Operations, i.e. the run-time production environment, risk management has the added focus on operational risks. While contractual issues between the corporate IT organization and the companies, business units or departments in the corporation that are the customers of the software product are typically not relevant on a product level, contracts with software providers need special attention. Dependent on the industry the corporation, company or business unit is doing business in there may be specific legal or regulatory requirements.

Product planning needs to be managed in close cooperation between the corporate IT organization and the companies, business units or departments in the corporation that are the customers of the software product. The business side of the enterprise architecture, in particular business process models and data models, play an important role with regard to requirements and integration aspects. In product life cycle management, profit considerations and market share are typically not relevant unless the IT organization is run as a profit center and/or the corporation allows its business units to work with external competitors.

The relevance of the strategic management aspects depends on how the business relationship between the corporate IT organization and the companies, business units or departments in the corporation is designed. If the corporate IT organization is run as a cost center its focus will be on innovation management and resource

management. The more the corporate IT organization is run as a business unit of its own, the more relevant the other strategic management aspects become.

On the orchestration side, marketing and sales are usually not relevant unless the corporation allows its business units to work with external competitors. An exception is customer relationship management with the company-internal departments and users as customers. Services usually include Operations, i.e. the run-time production environment which is governed by the IT service management processes. The product manager is in a monitoring role and may be directly involved in critical situations.

7.4 ISPMA

The International Software Product Management Association (ISPMA, www.ispma.org) is an open non-profit association of experts, companies, research institutes, and practitioners with the goal to foster software product management excellence across industries. ISPMA was started in 2009 and legally founded in 2011. As of November 2016, it has more than 800 members worldwide. Hans-Bernd Kittlaus is ISPMA's current chairman, Samuel Fricker is ISPMA's former chairman. Both have been founding board members of ISPMA since ISPMA's inception.

ISPMA aims at establishing software product management as a discipline of its own in both academia and industry, and disseminates and maintains a Curriculum and a Certifiable Body of Knowledge (SPMBoK). The SPMBoK is documented in syllabi that are the basis for training courses and certification exams:

- Foundation Level.
- Excellence Level: Product Strategy.
- Excellence Level: Product Planning.
- Excellence Level: Strategic Management.
- Excellence Level: Orchestration.

The foundation level is targetted at participants with up to 5 years of practical experience in the software area. They ought to have a fundamental understanding of the software business, but the training requires no specific technical or commercial competencies. The excellence level modules are targetted at product managers who already have the ISPMA Foundation Level Certificate or comparable SPM experience of at least 3 years.

ISPMA's results are applicable to the software industry, to vendors of software-intensive products and technical and human services in other industries (embedded software), and to corporate IT organizations in all industries.

The syllabi are available for free on the ISPMA web site. Training courses can be offered by commercial training providers and universities after approval by ISPMA. Certification exams are conducted by independent certification agencies that issue the certificates on behalf of ISPMA.

ISPMA also provides a platform for communication and exchange between its members, be it on conferences, in workshops and working groups, or on the internet.

There are different membership types:

- Fellow Member: Distinguished expert from industry or academia that is elected by the existing fellow members. Fellow members are committed to contribute to ISPMA work results and represent ISPMA.
- Certified Member: Practitioner or academic member who has at least one of ISPMA's certificates.
- Subscribing Member: People from industry and academia who are interested in SPM.
- Company Member: Company or academic institution which is committed to excellence in SPM and wants to support ISPMA. Company members nominate delegates who have the rights of fellow members.

ISPMA provides a lot of value to the SPM community, participants in trainings and certification exams, and companies and academic institutions interested in SPM that is unique in comparison to all the other players in product management education:

- Focus on software only.
- Tight cooperation between experts from industry and academia for continuous updates and improvements of the SPMBoK based on the latest developments in business, technology and methodology.
- Non-profit organization.
- Strict separation between
  - ISPMA as developer of curriculum, syllabi and exams (non-profit).
  - Training provider and trainers (commercial or academic).
  - Certification agencies.
- High quality of SPMBOK.
- High confidentiality of exam contents.
- High value of certificates (due to the separation described above).
- Free availability of syllabi.
- Frequent information on latest developments in SPM.
- Open platform for networking, exchange and cooperation.

With these elements, ISPMA is helping individuals to learn about SPM or improve their SPM skills. ISPMA also helps companies that want to establish or improve their SPM organizations. And ISPMA creates a basis for training providers and trainers who want to offer SPM trainings. All of these groups are welcome to become personal and company members of ISPMA.

Glossary

Brand Awareness A measure of whether a brand is correctly identified by potential customers. Typically expressed as a percentage of a target market. There are many different approaches to measuring this statistic, but it typically means that a customer can respond to a brand after viewing its logo or packaging.

Business Case A decision support and planning approach for comparing the costs and benefits associated with a proposed initiative.

Business Model Description of the rationale of how an organization creates, delivers and captures value by interacting with suppliers, employees, customers and partners.

Business Model Archetype A basic pattern of doing business. Available archetypes are creator, distributor, lessor and broker.

Category Maturity Life Cycle A model that describes the rise, duration, and decline of a category of product or service in terms of total revenue or number of users.

Channel A sequence of intermediaries through which goods and services as well as the compensation are transferred between a company and its customers.

Cloud Computing Service and delivery model for the provisioning of IT components through the internet based on an architecture that enables a high level of scalability and reliability.

Company Board The entity of a company which is responsible for the definition and communication of strategy, vision and mission to the rest of the company. Also, it has the managerial supervision of the different departments, including product management.

Competitor A competitor of company A is another company that sells products and/or services to A's target market (or a subset thereof) which are similar to A's products and/or services.

Constraint Business, project or design decisions taken in advance to ensure the solution fits business, managerial and contextual concerns. These decisions limit the solution space.

This glossary is aligned with ISPMA's Glossary 2.0 as of December 2016.

H.-B. Kittlaus, S.A. Fricker, *Software Product Management*,
DOI 10.1007/978-3-642-55140-6

Continuous Delivery Automated push of software into the production environment or delivery to customers.

Continuous Deployment Combination of continuous integration and continuous delivery to automatically deliver code changes to customers.

Continuous Integration Automated integration, build, and test of software in the development environment.

Conversion Rate Metric for the number of customers who have completed a transaction on a web site divided by the total number of website visitors.

Copyright A form of intellectual property right that gives the author of an original work exclusive rights for publishing, distributing and adapting the work.

Corporate Strategy The basic long-term goals of an enterprise and the courses of action for carrying out these goals.

Cost Per Lead Metric for the average amount of money invested to acquire a new lead.

Cost Structure The types and relative proportions of fixed and variable costs connected to a business model.

Customer Segment A subset of existing and/or potential customers targeted by a common value proposition.

Customer A party that receives or consumes products and/or services from a second party.

Delivery Model A description of the mechanisms in which a product is made available to customers. Examples: Licensed product vs. Software-as-a-Service (SaaS).

DevOps Tight cooperation of Development and Operations with the objective to significantly increase the speed and quality of software deployment.

Embedded Software Software parts of software-intensive systems that are not marketed and priced as separate entities.

Functional Requirement A statement that identifies what a product or process must accomplish to produce required behavior and/or results.

Functional Support Plan Describes the activities, deliverables, budgets, dependencies, and schedules for a business function to members of a cross-functional product team, on behalf of a product.

Innovation Management The discipline of managing processes related to innovation. Innovation management allows the organization to respond to external or internal opportunities, and use its creative efforts to introduce new ideas, processes or products.

Intellectual Property Exclusive rights that are granted by law to the owner(s) of intangible assets that result from creations of the mind, such as inventions, literary and artistic works, and symbols, names, and images used in commerce.

Key Performance Indicator (KPI) A specific numerical measure that represents the progress towards a strategic goal, objective, output, activity, or further input.

Kano Analysis A technique for understanding which product features will help drive customer satisfaction.

License A set of rights concerning a licensor's intellectual property which a licensor grants to a licensee.

License Agreement A legal document that describes a license and the related financial conditions.

Market (a) The area of economic activity in which buyers and sellers of goods and services come together, and the forces of supply and demand affect prices. (b) A geographic area of demand for commodities or services.(c) A specified category of potential buyers.

Market Analysis Analysis of all aspects relevant for a particular market in its current state and over the strategic time frame including market structure, competitors, market shares, customer preferences and behavior.

Market Segmentation Division of a market into sub-sets called market segments that are distinct from each other, and homogeneous with regard to certain criteria.

Market Share Percentage of revenue or volume that a particular player makes in a particular market or market segment in relation to the market's total revenue or volume.

Marketing (a) The activities that are involved in making people aware of a company's products and making sure that the products are available to be bought.(b) The organizational and/or functional unit in an organization that is responsible for (a).

Marketing Communication (MarCom) Set of marketing activities that concerns executing the marketing strategy to create communication deliverables including advertising, branding, graphic design, promotion, publicity, public relations and more.

Matrix Organization Organizational structure in which individuals report to more than one person

Minimum Viable Product The minimum feature set of a new product that is derived through a learning phase and that some customers are willing to pay for in the first release.

Mission Statement Definition of the present activities or purpose of an organization by saying what it does for whom and how.

Net Promoter Score Method to measure customer loyalty by asking a single 0–10 scale question: "How likely is it that you would recommend $BRAND$ to a friend or colleague?". Your promoter score is the percentage of customers who answer 9 or 10 subtracted by those who answer 0–6.

Non-functional Requirement A requirement that pertains to a quality concern that is not covered by functional requirements. Also referred to as -> Quality requirement

Open Source Software Software that can be freely accessed, used, changed, and shared (in modified or unmodified form) by anyone. Open source software is made by one or more people, and distributed under licenses that comply with the Open Source Definition.

Partner A party who joins one or more other parties based on an agreement that defines the terms and conditions of the relationship (partnership).

Patent A patent is an exclusive intellectual property right for an invention granted to the inventor for a defined timeframe by an authorized body of a sovereign state. The right is granted for the territory of that sovereign state. It is the right to exclude others from making, using, offering for sale, or selling the invention. The types of inventions covered by patent law can be different from state to state.

Performance Management Continuous tracking and analysis of selected KPIs relevant for business success, plus timely action taking if needed

Pricing All activities required to set, communicate, and negotiate prices in a convincing way.

Process Improvement All activities to analyze, plan and execute changes in processes with the goal to optimize the defined process KPIs.

Process Model An abstract description of one or more processes. A process model typically describes a process as a sequence of activities and the involved roles and responsibilities.

Product A combination of goods and services, which a supplier/development organization combines in support of its commercial interests to transfer defined rights to a customer.

Product Analysis Analysis of all business aspects relevant for a particular product in its current state and over the strategic time frame including KPIs like revenue, revenue distribution, footprints, and market shares.

Product Family A group of software products that are marketed as belonging together under a common family name.

Product Life Cycle The evolution of a product from its conception to its discontinuance and market withdrawal.

Product Life Cycle Management The management of the business and technical aspects of a software product with regard to its position in its life cycle.

Product Line A set of products based on a common platform with defined (static or dynamic) variability tailored to different markets and users.

Product Management (a) The discipline which governs a product along the product life cycle with the objective to generate the biggest possible value to the business.(b) The organizational and/or functional unit in an organization that is responsible for (a).

Product Manager A person responsible for -> Product Management in an organization. An organization can have multiple product managers.

Product Marketing Applying -> Marketing to a -> Product. Translates strategic marketing decisions to the product level.

Product Platform The technical foundation on which several software products are based.

Product Portfolio Set of products or services offered by a company.

Product Portfolio Management The activity of making decisions about investments in the products included in the product portfolio over the strategic timeframe.

Product Roadmap A document that provides features or themes of the product releases to come over the strategic timeframe. The creation of a roadmap is influenced by the product strategy designed for this product.

Product Scope Abstract description of the functional and quality characteristics of the product.

Product Strategy (a) Combination of the strategic goals and measures for the product, i.e. aspects that need to be defined and managed for the strategic timeframe of the product. See corresponding column in ISPMA's SPM Framework. (b) Consistent documentation containing the following items and their evolution during the strategic timeframe: • Product vision• Product definition• Target market, potential segments• Delivery model• Product positioning• Sourcing• Business plan• Roadmap

Product Vision Conceptual description of what the future product will be at the end of the strategic timeframe, i.e. high-level descriptions of a product concept and a corresponding business model.

Product-Technology Roadmap Overview of the relationship between product releases (product evolvement) and successive technology generations.

Quality Requirement A requirement that pertains to a quality concern that is not covered by functional requirements. Also referred to as -> Non-functional requirement.

Release (a) Product release: an instance of the product that is delivered to customers, and maintained as part of product maintenance.(b) Pre-release: a result of development activity that is testable, e.g. the result of a sprint in Scrum.

Release Definition The result of selecting the requirements to be implemented in the next release. Usually this result is documented including statements about the relationship between the selected requirements and strategic objectives.

Release Planning The process of selecting the requirements for the next release.

Requirement (a) A condition or capability needed to solve a problem or achieve an objective.(b) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.(c) A documented representation of a condition or capability as in definition (a) or (b). Three different types of requirements are distinguished: functional requirements, quality requirements and constraints.

Requirements Engineering (a) The disciplined and systematic approach (i.e., "engineering") for elicitation, documentation, analysis, agreement, verification, and management of requirements while considering market, technical, and economic goals.(b) Activity within systems engineering and software engineering.

Requirements Management Planning, executing, monitoring, and controlling any or all of the work associated with requirements elicitation and collaboration, requirements analysis and design, and requirements life cycle management.

Requirements Prioritization The activity during which the most important requirements for the product are determined. As priorities change over time this activity is often targeted at the next release of the product.

Requirements Triage An activity for early and fast acceptance/rejection of requirements.

Resource Management The efficient and effective development of an organization's resources. In the software business resources are primarily people, existing software and systems that the software runs on or is developed on.

Return On Investment Metric for the average amount of revenue divided by the related cost.

Revenue Money collected by an organization in return for products and/or services.

Revenue Model Set of all -> revenue streams of a company.

Revenue Stream Describes generation of compensation over time for a product, service or company as revenue or in non-monetary ways. Non-monetary aspects may be data or services in return.

Risk Management The identification, assessment, and prioritization of risks followed by coordinated and economical application of resources to minimize, monitor, and control the probability and/or impact of unfortunate and/or undesired events.

Scenario Description of a real or imagined situation under a defined set of assumptions.

Service (a) Useful labor that does not produce a tangible commodity (as in "professional services").(b) A provision for maintenance and repair (as in "software maintenance service").(c) The technical provision of a function through a software component that can be accessed by another software component, often over a network and executed on a remote server (as in "web services" or "Software-as-a-Service").

Service Level Agreement (SLA) Agreement between two or more parties about the target values a service-giving party has to achieve for the defined measures that are relevant for quality and cost of the service.

Software Ecosystem A network of people and/or companies that forms around a software vendor or a product or product platform. The relationships in this network have the goal to achieve benefits for all participants and can be formalized or not. Formalized relationships are called partnerships.

Software Intensive System A system where a significant part of the value originates from software.

Software Product A product whose primary component is software.

Software Product Family A group of software products which for marketing reasons are marketed as belonging together under a common family name.

Software Product Line A set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission, and that are developed from a common set of core assets in a prescribed way.

Software Product Management The management of a software product or the software components of a software-intensive product over its life cycle with the objective of generating the biggest possible value to the business.

Software Product Management Competence Model Competence model that guides product management in process improvement.

Software Product Manager Product manager of a software product or the software components of a software-intensive product.

Software Value Map A decomposition of the "value" concept that details value of a software intensive product from the main areas of financial, customer, internal business process, and innovation and learning perspectives.

Software-as-a-Service (SaaS) A delivery model for software that is used in cloud computing.

Solution (a) A product that is a combination of other products, human services, and possibly some glue code and customization.(b) A combination of products and customer-specific code that is developed and implemented for a specific customer.

Sourcing The process of ensuring that all required resources are available when they are needed.

Stakeholder A person, group, or organization that has direct or indirect stake in an organization because it can affect or be affected by the organization's actions, objectives, and policies.

Strategic Marketing The way a firm effectively differentiates itself from its competitors by effectively segmenting the market, selecting the appropriate targets and consistently developing a better value positioning to customers than its competitors.

Target Market A set of market segments to which a particular product is marketed to.

Trade Secret Something which has economic value to a business because it is not generally known or easily discoverable by observation such as an algorithm and for which efforts have been made to maintain secrecy.

Trademark A distinctive identifier, such as a phrase, word or sign, for certain products or services as those produced or provided by a specific person or enterprise. Protection of trademarks depends on local law.

User A person or thing that uses something i.e. products or services.

User Experience Every aspect of the users' interactions with a software product or component with the purpose of shaping the user's behaviors, attitudes, and emotions about that product or component.

Value Proposition Description of the benefits customers can expect from one product, or from the products and services of a company.

Virtual Team A group of individuals who work together across time, space and organizational boundaries with links strengthened by webs of communication technology.

Bibliography

[Aaker13] Aaker, D.A.: Strategic Market Management. Wiley, New York (2013)

[AdnKap10] Adner, R., Kapoor, R.: Value creation in innovation ecosystems: how the structure of technological interdependence affects firm performance in new technology generations. Strateg. Manag. J. **31**(3), 306–333 (2010)

[AlbKap03] Albright, R., Kappel, T.: Roadmapping the corporation. Research-Technology Management. **46**(2), 31–40 (2003)

[Allen06] Allen, P., Higgins, S.: Service Orientation: Winning Strategies and Best Practices. Cambridge University Press, Cambridge (2006)

[AlmSenBl16] Almquist, E., Senior, J., Bloch, N.: The elements of value. Harv. Bus. Rev. **94**, 46–53 (2016)

[Alvarez14] Alvarez, C.: Lean Customer Development—Build Products Your Customers Will Buy. O'Reilly, Sebastopol, CA (2014)

[AndNar98] Anderson, J.C., Narus, J.A.: Business marketing: understand what customers value. Harv. Bus. Rev. **76**, 53–65 (1998)

[AndZei84] Anderson, C., Zeithaml, C.: Stage of the product life cycle, business strategy, and business performance. Acad. Manag. J. **27**(1), 5–24 (1984)

[Arthur96] Arthur, W.B.: Increasing returns and the new world of business. Harv. Bus. Rev. **74**, 100–109 (1996)

[ArWeBriFi10] Artz, P., van de Weerd, I., Brinkkemper, S., Fieggen, J.: Productization: Transforming from Developing Customer-Specific Software to Product Software. International conference on software business (ICSOB 2010), Jyväskylä, Finland, 2010

[AveBePe15] Avedillo, J.G., Begonha, D., Peyracchia, A.: Two Ways to Modernize IT Systems for the Digital Era. McKinsey Insights, Spain (2015)

[Axelos16] Axelos: ITIL Practitioner Guide. TSO, Norwich (2016)

[BasWebZh15] Bass, L., Weber, I., Zhu, L.: DevOps: A Software Architect's Perspective. Addison-Wesley, Upper Saddle River (2015)

[Bech15] Bech, H.P.: Building Successful Partner Channels. TBK Publishing, Klampenborg (2015)

[BekWeer10] Bekkers, W., van de Weerd, I.: SPM Maturity Matrix. Technical report UU-CS-2010-013, University of Utrecht (2010)

[BVSB10] Bekkers, W., van de Weerd, I., Spruit, M., Brinkkemper, S.: A Framework for Process Improvement in Software Product Management. European conference on software process improvement (EuroSPI 2010), Grenoble, France, 2010

[BenMcF13] Benko, C.A., McFarlan, W.: Connecting the Dots. Aligning Your Project Portfolio with Corporate Objectives. McGraw-Hill, London (2013)

[BerAnd05] Berander, P., Andrews, A.: Requirements prioritization. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements. Springer, Berlin (2005)

H.-B. Kittlaus, S.A. Fricker, *Software Product Management*,
DOI 10.1007/978-3-642-55140-6

[BGRTSF11] Berntsson-Svensson, R., Gorschek, T., Regnell, B., Torkar, R., Shahrokni, A., Feldt, R.: Quality requirements in industrial practice—an extended interview study at eleven companies. IEEE Trans. Softw. Eng. **38**(4), 923–935 (2011)

[Besaha03] Besaha, B.: Bounty hunting in the patent base. Commun. ACM. **46**(3), 27–29 (2003)

[Biering04] Biering, S.: Preis- und Produktstrategien für digitale Produkte, untersucht am Beispiel des Software-Marktes (Dissertationsschrift). Haufe, Freiburg (2004)

[Blank13a] Blank, S.: The Four Steps to the Epiphany, 2nd edn. K & S Ranch, Menlo Park (2013)

[Blank13b] Blank, S.: A new way to look at competitors. https://steveblank.com/2013/11/08/a-new-way-to-look-at-competitors/ (2013). Accessed 25 Dec 2016

[Blank13c] Blank, S.: Why the lean startup changes everything. Harv. Bus. Rev. **91**(5), 63–72 (2013)

[BlanDorf12] Blank, S., Dorf, B.: The Startup Owner's Manual: The Step-By-Step Guide for Building a Great Company. K & S Ranch, Menlo Park (2012)

[Blank10] Blank, S.: Perfection by subtraction—the minimum feature set. http://steveblank.com/2010/03/04/perfection-by-subtraction-the-minimum-feature-set/ (2010). Accessed 25 Dec 2016

[BoGiRo06] Bonaccorsi, A., Giannangeli, S., Rossi, C.: Entry strategies under competing standards: hybrid business models in the open source software industry. Manag Sci. **52**(7), 1085–1098 (2006)

[Bosch09] Bosch, J.: From Software Product Lines to Software Ecosystems. 13th international software product line conference (SPLC 2009), San Francisco, CA, USA, 2009

[Bosch12] Bosch, J.: Building Products as Innovation Experiment Systems. International conference on software business (ICSOB 2012), Cambridge, MA, USA, 2012

[Boulding62] Boulding, K.: Conflict and Defense: A General Theory. Harper & Brothers, New York (1962)

[Brown08] Brown, T.: Design thinking. Harv. Bus. Rev. **86**(6), 84–92 (2008)

[Brynjo03] Brynjolfsson, E.: The IT productivity gap. Optimize Mag. **21**, 26–43 (2003)

[BrMaJaHe96] Bruckhaus, T., Madhavji, N., Janssen, I., Henshaw, J.: The impact of tools on software productivity. IEEE Softw. **13**(5), 29–38 (1996)

[BSA16] BSA: Seizing opportunity through license compliance: BSA global software survey. http://globalstudy.bsa.org/2016/downloads/studies/BSA_GSS_US.pdf (2016). Accessed 25 Dec 2016

[BuStTh10] Burke, A., van Stel, A., Thurik, R.: Blue ocean vs. five forces. Harv. Bus. Rev. **88**(5), 28–29 (2010)

[BusZim12] Buse, R.P.L., Zimmermann, T.: Information Needs for Software Development Analytics. Proceedings of the 34th international conference on software engineering (ICSE 2012 SEIP Track), Zurich, Switzerland, 2012

[Butje05] Butje, M.: Product Marketing for Technology Companies. Butterworth-Heinemann, Burlington (2005)

[BuDiHe12] Buxmann, P., Diefenbach, H., Hess, T.: The Software Industry. Springer, Heidelberg (2012)

[BuJaPo11] Buxmann, P., Jansen, S., Popp, K.M.: The Sun also Sets: Ending the Life of a Software Product. International conference on software business (ICSOB 2011), Brussels, Belgium, 2011

[Cagan08] Cagan, M.: Inspired—How to Create Products Customers Love. SVPG Press, Sunnyvale, CA (2008)

[Carey09] Carey, P.: Data Protection: A Practical Guide to UK and EU Law. Oxford University Press, Oxford (2009)

[Carlshamre02] Carlshamre, P.: Release planning in market-driven development: provoking an understanding. Requir. Eng. **7**(3), 139–151 (2002)

[CSLRH01] Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., Natt och Dag, J.: An Industrial Survey of Requirements Interdependencies in Software Product

Release Planning. 5th international symposium on requirements engineering (RE'01), Toronto, Canada, 2001

[CheCheMe13] Chen, S., Cheng, A., Mehta, K.: A review of telemedicine business models. Telemed. e-Health. **19**(4), 287–297 (2013)

[Chesbrou05] Chesbrough, H.W.: Open Innovation: The New Imperative for Creating and Profiting from Technology. Harvard Business Review Press, Boston (2005)

[Christen13] Christensen, C.A.: The Innovator's Solution. Harvard Business Review Press, Boston (2013)

[ClaHeyScho08] Classen, A., Heymans, P., Schobbens, P.: What's in a Feature: A Requirements Engineering Perspective. 11th international conference on fundamental approaches to software engineering (FASE 2008), Budapest, Hungary, 2008

[ClSeRBC07] Cleland-Huang, J., Settimi, R., Romanova, E., Berenbach, B., Clark, S.: Best practices for automated traceability. IEEE Comput. **40**(6), 27–35 (2007)

[Cohn04] Cohn, M.: User Stories Applied: For Agile Software Development. Mountain Goat Software, Denver, CO (2004)

[Cohn06] Cohn, M.: Agile Estimating and Planning. Prentice Hall PTR, Upper Saddle River (2006)

[CollThom02] Collberg, C.S., Thomborson, C.: Watermarking, tamper-proofing, and obfuscation—tools for software protection. IEEE Trans. Softw. Eng. **28**(8), 735–746 (2002)

[Condon02] Condon, D.: Software Product Management: Managing Software Development from Idea to Product to Marketing to Sales. Aspatore Books, Boston (2002)

[CooEdg09] Cooper, R.G., Edgett, S.J.: Generating Breakthrough New Product Ideas: Feeding the Innovation Funnel. Product Development Institute, Canada (2009)

[CooEdg08] Cooper, R.G., Edgett S.J.: Ideation for product innovation: what are the best methods? PDMA Vis. Mag. **32**, 12–17 (2008)

[CoEdKl01] Cooper, R.G., Edgett, S.J., Kleinschmidt, E.J.: Portfolio Management for New Products, 2nd edn. Perseus Books, Cambridge (2001)

[Cooper00] Cooper, R.G.: Product Leadership—Creating and Launching Superior New Products. Perseus Books, Cambridge (2000)

[Coulter12] Coulter, M.: Strategic Management in Action, 6th edn. Pearson Prentice Hall, Upper Saddle River (2012)

[CrolYosk13] Croll, A., Yoskovitz, B.: Lean Analytics—Use Data to Build a Better Startup Faster. O'Reilly, Sebastopol, CA (2013)

[Cusuma07] Cusumano, M.: The changing Labyrinth of software pricing. Commun. ACM. **50**(7), 19–22 (2007)

[Cusuma04] Cusumano, M.: The Business of Software. Free Press, New York (2004)

[Cusuma03] Cusumano, M.: Finding your balance in the products and services debate. Commun. ACM. **46**(3), 15–17 (2003)

[DavBrHo07] Daviesa, A., Bradyb, T., Hobday, M.: Organizing for solutions: systems seller vs. systems integrator. Ind. Mark. Manag. **36**(2), 183–184 (2007)

[Davis05] Davis, A.: Just Enough Requirements Management. Dorset House Publishing, New York (2005)

[DeLaat05] De Laat, P.B.: Copyright or copyleft? An analysis of property regimes for software development. Res Policy. **34**(10), 1511–1532 (2005)

[DemLec06] Demil, B., Lecocq, X.: Neither market nor hierarchy nor network: the emergence of bazaar governance. Organ. Stud. **27**(10), 1447–1466 (2006)

[DenCle04] Denne, M., Cleland-Huang, J.: Software by Numbers: Low-Risk, High-Return Development. Prentice Hall PTR, Upper Saddle River (2004)

[DenLew17] Denning, P.J., Lewis, T.G.: Exponential laws of computing growth. Commun. ACM. **60**(1), 54–65 (2017)

[DenDun03] Denning, P.J., Dunham, R.: The missing customer. Commun. ACM. **46**(3), 19–23 (2003)

[Druck73] Drucker, P.: Management: Tasks, Responsibilities, Practices, pp. 64–65. Harper and Row, New York (1973)

[Ebert07] Ebert, C.: The impacts of software product management. J. Syst. Softw. **80**(6), 850–861 (2007)

[Ebert09] Ebert, C.: Software product management. CrossTalk. **16**, 15–19 (2009)

[Ebert11] Ebert, C.: Global Software and IT: A Guide to Distributed Development, Projects, and Outsourcing. Wiley, Hoboken (2011)

[EberBrin14] Ebert, C., Brinkkemper, S.: Software product management—an industry evaluation. J. Syst. Softw. **95**, 10–18 (2014)

[EberDumk07] Ebert, C., Dumke, R.: Software Measurement: Establish—Extract—Evaluate—Execute. Springer, Berlin (2007)

[Fagerb05] Fagerberg, J.: Innovation—a guide to the literature. In: Fagerberg, J., Mowery, D., Nelson, R. (eds.) The Oxford Handbook of Innovation. Oxford University Press, Oxford (2005)

[Farmer06] Farmer, E.: The Gatekeeper's guide, or how to kill a tool. IEEE Softw. **23**(6), 12–13 (2006)

[FeFiHL05] Feller, J., Fitzgerald, B., Hissam, S.A., Lakhani, K.R. (eds.): Perspectives on Free and Open Source Software. MIT Press, Cambridge (2005)

[FennRask08] Fenn, J., Raskino, M.: Mastering the Hype Cycle: How to Choose the Right Innovation at the Right Time. Harvard Business Review Press, Boston (2008)

[FiUrPa11] Fisher, R., Ury, W.L., Patton, B.: Getting to Yes: Negotiating Agreement Without Giving in, upd. rev. edn. Penguin, New York (2011)

[FleBen15] Fleisher, C., Bensoussan, B.: Business and Competitive Analysis: Effective Application of New and Classic Methods, 2nd edn. Pearson Education, Upper Saddle River (2015)

[FleBen02] Fleisher, C., Bensoussan, B.: Strategic and Competitive Analysis: Methods and Techniques for Analyzing Business Competition. Prentice Hall, Upper Saddle River (2002)

[FloyWool94] Floyd, S.W., Wooldridge, B.: Dinosaurs or dynamos? Recognizing middle management's strategic role. Acad. Manag. Exec. **8**(4), 47–57 (1994)

[FlyvBud11] Flyvbjerg, B., Budzier, A.: Why your IT project may be riskier than you think. Harv. Bus. Rev. **89**(9), 23–25 (2011)

[FotrFric16] Fotrousi, F., Fricker, S.: Software Analytics for Planning Product Evolution. International conference on software business (ICSOB 2016), Ljubljana, Slovenia, 2016

[FoFrFi14] Fotrousi, F., Fricker, S., Fiedler, M.: Quality Requirements Elicitation based on Inquiry of Quality-Impact Relationships. IEEE 22nd international requirements engineering conference (RE 14), Karlskrona, Sweden, 2014

[FoFrFiLG14] Fotrousi, F., Fricker, S., Fiedler, M., Le Gall, F.: KPIs for Software Ecosystems: A Systematic Mapping Study. International conference on the software business (ICSOB 2014), Paphos, Cyprus, 2014

[FrenRav59] French, J.R.P., Raven, B.: The bases of social power. In: Cartwright, D.P. (ed.) Studies in Social Power, pp. 150–167. University of Michigan Press, Ann Arbor (1959)

[Fricker12] Fricker, S.A.: Software product management. In: Maedche, A., Botzenhardt, A., Neer, L. (eds.) Software for People—Fundamentals, Trends and Best Practices, pp. 53–81. Springer, Heidelberg (2012)

[Fricker14] Fricker, S.A., Grau, R., Zwingli, A.: Requirements engineering: best practice. In: Fricker, S.A., Thümmler, C., Gavras, A. (eds.) Requirements Engineering for Digital Health. Springer, Cham (2014)

[FricGru08] Fricker, S.A., Grünbacher, P.: Negotiation Constellations—Method Selection Framework for Requirements Negotiation. International working conference on requirements engineering: foundation for software quality (RefsQ 2008), Montpellier, France, 2008

[FricSFT16] Fricker, S.A., Schneider, K., Fotrousi, F., Thümmler, C.: Workshop videos for requirements communication. Requir. Eng. **21**(4), 521–552 (2016)

[FricSchu12] Fricker, S.A., Schumacher, S.: Release Planning with Feature Trees: Industrial Case. 18th international working conference on requirements engineering: foundation for software quality (RefsQ 2012), Essen, Germany, 2012

[FrGoBySc10] Fricker, S.A., Gorschek, T., Byman, C., Schmidle, A.: Handshaking with implementation proposals: negotiating requirements understanding. IEEE Softw. **27**(2), 72–80 (2010)

[Gartner16] Gartner Group: Gartner Market Databook, 2Q16 Update (2016)

[Gartner13] Gartner Group: Understanding Gartner's Hype Cycles. Gartner Group, Stamford, ID Number G00251964 (2013)

[GausWein89] Gause, D.C., Weinberg, G.M.: Exploring Requirements: Quality Before Design. Dorset House, New York (1989)

[Glinz08] Glinz, M.: A risk-based, value-oriented approach to quality requirements. IEEE Soft. **25**(2), 34–41 (2008)

[GopSand97] Gopal, R.D., Sanders, G.L.: Preventive and deterrent controls for software piracy. J. Manag. Inf. Syst. **13**(4), 29–47 (1997)

[GoFrPaKu10] Gorschek, T., Fricker, S.A., Palm, K., Kunsman, S.: A lightweight innovation process for software-intensive product development. IEEE Softw. **27**(1), 37–45 (2010)

[GorsWohl06] Gorschek, T., Wohlin, C.: Requirements abstraction model. Requir. Eng. **11**(1), 79–101 (2006)

[GoWoGL06] Gorschek, T., Wohlin, C., Garre, P., Larsson, S.: A model for technology transfer in practice. IEEE Softw. **23**(6), 88–95 (2006)

[Gorche11] Gorchels, L.: The Product Manager's Handbook: The Complete Product Management Resource, 4th edn. McGraw Hill, New York (2011)

[GotSei13] Gothelf, J., Seiden, J.: Lean UX—Applying Lean Principles to Improve User Experience. O'Reilly, Sebastopol, CA (2013)

[GottGor12] Gottesdiener, E., Gorman, M.: Discover to Deliver: Agile Product Planning and Analysis. EBG Consulting, Acton, MA (2012)

[GreRuh04] Greer, D., Ruhe, G.: Software release planning: an evolutionary and iterative approach. Inf. Softw. Technol. **46**(4), 243–253 (2004)

[GriHau96] Griffin, A., Hauser, J.R.: Integrating R&D and marketing: a review and analysis of the literature. J. Prod. Innov. Manag. **13**(3), 191–215 (1996)

[Groenv97] Groenveld, P.: Roadmapping integrates business and technology. Res. Technol. Manag. **40**(5), 49–58 (1997)

[Haines14] Haines, S.: The Product Manager's Desk Reference, 2nd edn. McGraw Hill, New York (2014)

[Hall13] Hall, K.: Making the Matrix Work—How Matrix Managers Engage People and Cut Through Complexity. Nicholas Brealey Publishing, London (2013)

[Harmon14] Harmon, P.: Business Process Change. Morgan Kaufmann, Amsterdam (2014)

[HassTrac06] Hassenzahl, M., Tractinsky, N.: User experience—a research agenda. Behav. Inf. Technol. **25**(2), 91–97 (2006)

[Hawkins08] Hawkins, S.: Magic Quadrants and Market Scopes: How Gartner Evaluates Vendors Within a Market, Gartner, Inc., Research Document G00154752, January 2008

[Hecker99] Hecker, F.: Setting up shop: the business of open-source software. IEEE Softw. **16**(1), 45–51 (1999)

[Hender70] Henderson, B.: The Product Portfolio. Boston Consulting Group, Boston (1970)

[HerrDan08] Herrmann, A., Daneva, M.: Requirements Prioritization based on Benefit and Cost Prediction: An Agenda for Future Research. 16th IEEE international conference on requirements engineering (RE'08), Barcelona, Spain, 2008

[Herzwurm10] Herzwurm, G.: Produktmanagement in der IT: Geschäftsmodelle und Produktpositionierung, 7. Fachtagung Software Management—Vom Projekt zum Produkt (FTSWM'2010), Aachen, Germany, 2010

[HerzPiet09] Herzwurm, G., Pietsch, W.: Management von IT-Produkten—Geschäftsmodelle, Leitlinien und Werkzeugkasten für softwareintensive Systeme und Dienstleistungen. Springer, Heidelberg (2009)

[HerzPiet08a] Herzwurm, G., Pietsch, W.: Guidelines for the Analysis of IT Business Models and Strategic Positioning of IT-Products. 2nd international workshop on software product management (IWSPM'2008), Barcelona, Spain, 2008

[HerzPiet08b] Herzwurm, G., Pietsch, W.: Management von IT-Produkten. Geschäftsmodelle, Leitlinien und Werkzeugkasten für softwareintensive Systeme und Dienstleistungen. Springer, Heidelberg (2008)

[HiAlCeBa13] Hillenbrand, P., Alcauter, S., Cervantes, J., Barrios, F.: Better branding: brand names can influence consumer choice. J. Prod. Brand Manag. **2**(4), 300–308 (2013)

[HoRoPL00] Hoch, D.J., Roeding, C.R., Purkert, G., Lindner, S.K.: Secrets of Software Success. Harvard Business School Press, Boston (2000)

[HooFar01] Hooks, I.F., Farry, K.A.: Customer-Centered Products—Creating Successful Products through Smart Requirements Management. Amacom, New York (2001)

[HumMol11] Humble, J., Molesky, J.: Why enterprises must adopt DevOps to enable continuous delivery. Cutter IT J. **24**(8), (2011)

[HumFar10] Humble, J., Farley, D.: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, Upper Saddle River (2010)

[IanLev04a] Iansiti, M., Levien, R.: Strategy as ecology. Harv. Bus. Rev. **82**(3), 68–81 (2004)

[IanLev04] Iansiti, M., Levien, R.: The Keystone Advantage—What the New Dynamics of Business Ecosystems Mean for Strategy, Innovation, and Sustainability. Harvard Business School Press, Boston (2004)

[IDC16] IDC: Worldwide Software as a Service and Cloud Software Forecast, 2016–2020. IDC, Framingham (2016)

[ISO98] ISO/AWI TR 9241-1: Ergonomics of Human-System Interaction. Part 1: Introduction to the ISO 9241 Series (1998)

[ISPMA16] ISPMA.: www.ispma.org (2016). Accessed 25 Dec 2016

[JanBrCus13] Jansen, S., Brinkkemper, S., Cusumano, M.A. (eds.): Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry. Edward Elgar Publishing, Cheltenham (2013)

[JohGus00] Johnson, M.D., Gustafsson, A.: Improving Customer Satisfaction, Loyalty, and Profit: An Integrated Measurement and Management System. Jossey-Bass, San Francisco, CA (2000)

[Jones08] Jones, C.: Applied Software Measurement: Global Analysis of Productivity and Quality. McGraw-Hill, London (2008)

[Kappel01] Kappel, T.: Perspectives on roadmaps: how organizations talk about the future. Journal of Product Innovation Management. **18**(1), 39–50 (2001)

[KaSeTT84] Kano, N., Seraku, N., Takahashi, F., Tsuji, S.: Attractive quality and must-be quality. J. Jpn. Soc. Qual. Control. **14**(2), 147–156 (1984)

[KapNor96a] Kaplan, R.S., Norton, D.P.: Linking the balanced scorecard to strategy. Calif. Manage. Rev. **39**(1), 53–79 (1996)

[KapNor96b] Kaplan, R.S., Norton, D.P.: Using the balanced scorecard as a strategic management system. Harv. Bus. Rev. **74**(1), 75–85 (1996)

[KarlRyan97] Karlsson, J., Ryan, K.: A cost-value approach for prioritizing requirements. IEEE Softw. **14**(5), 67–74 (1997)

[Karlsson06] Karlsson, L.: Requirements prioritisation and retrospective analysis for release planning process improvement. Ph.D. Thesis at Lund University (2006)

[KaReTh06] Karlsson, L., Regnell, B., Thelin, T.: Case studies in process improvement through retrospective analysis of release planning decisions. Int. J. Softw. Eng. Knowl. Eng. **16**(06), 885–915 (2006)

[KarTRBW06] Karlsson, L., Thelin, T., Regnell, B., Berander, P., Wohline, C.: Pair-wise comparisons versus planning game partitioning—experiments on requirements prioritisation techniques. Empir. Softw. Eng. **12**(1), 3–33 (2007)

[KatSha85] Katz, M.L., Shapiro, C.: Network externalities, competition, and compatibility. Am. Econ. Rev. **75**(3), 424–440 (1985)

[KhuFrGor15] Khurum, M., Fricker, S., Gorschek, T.: The contextual nature of innovation—an empirical investigation of three software intensive products. Inf. Softw. Technol. **57**(1), 595–613 (2015)

[KhuGorW13] Khurum, M., Gorschek, T., Wilson, M.: The software value map—an exhaustive collection of value aspects for the development of software intensive products. J. Softw. Evol. Process. **25**(7), 711–741 (2013)

[KhuKhuGo07] Khurum, M., Khurum, A., Gorschek, T.: A Model for Early Requirements Triage and Selection (MERTS) Utilizing Product Line Strategies. 11th international software product line conference (SPLC 2007), Kyoto, Japan, 2007

[KimBeSp14] Kim, G., Behr, K., Spafford, G.: The Phoenix Project. IT Revolution Press, Portland (2014)

[KimMaub15] Kim, W.C., Mauborgne, R.: Blue Ocean Strategy: How to Create Uncontested Market Space and Make the Competition Irrelevant—Extended Ed. Harvard Business School Press, Boston (2015)

[Kittlaus15] Kittlaus, H.-B.: One Size Does Not Fit All: Software Product Management for Speedboats vs. Cruiseships. International conference on software business (ICSOB 2015), Braga, Portugal, 2015

[Kittlaus14] Kittlaus, H.-B.: Geschäftsmodelle. In: Hilber, M. (ed.) Handbuch Cloud Computing, pp. 29–53. Dr. Otto Schmidt Verlag, Köln (2014)

[Kittlaus12] Kittlaus, H.-B.: Software product management and agile software development: conflicts and solutions. In: Maedche, A., Botzenhardt, A., Neer, L. (eds.) Software for People—Fundamentals, Trends and Best Practices, pp. 83–96. Springer, Berlin (2012)

[KittClou09] Kittlaus, H.-B., Clough, P.: Software Product Management and Pricing—Key Success Factors for Software Organizations. Springer, Berlin (2009)

[KiRaSch04] Kittlaus, H.-B., Rau, C., Schulz, J.: Software-Produkt-Management—Nachhaltiger Erfolgsfaktor bei Herstellern und Anwendern. Springer, Berlin (2004)

[Klemens06] Klemens, B.: Math You Can't Use—Patents, Copyright, and Software. Brookings, Washington, DC (2006)

[KnasLeff17] Knaster, R., Leffingwell, D.: SAFe® 4.0 Distilled: Applying the Scaled Agile Framework® for Lean Software and Systems Engineering. Addison-Wesley, Upper Saddle River (2017)

[Kolb14] Kolb, D.: Experiential Learning: Experience as the Source of Learning and Development. Pearson Education, Upper Saddle River (2014)

[Kollm16] Kollmann, T.: E-Business, 6th edn. Springer, Berlin (2016)

[KKTLD15] Komssi, M., Kauppinen, M., Töhönen, H., Lehtola, L., Davis, A.: Roadmapping problems in practice: value creation from the perspective of the customers. Requir. Eng. **20**(1), 45–69 (2015)

[KostScha01] Kostoff, R., Schaller, R.: Science and technology roadmaps. IEEE Trans. Eng. Manag. **48**(2), 132–143 (2001)

[KotArm15] Kotler, P., Armstrong, G.: Principles of Marketing, 16th edn. Prentice Hall, Upper Saddle River (2015)

[Kruchten96] Kruchten, P.: A rational development process. CrossTalk. **9**(7), 11–16 (1996)

[KruCas14] Krüger, R., Casey, M.: Focus Groups: A Practical Guide for Applied Research, 5th edn. SAGE Publications, Thousand Oaks, CA (2014)

[Kude12] Kude, T.: The Coordination of Inter-Organizational Networks in the Enterprise Software Industry: The Perspective of Complementors. Peter Lang Verlag, Frankfurt (2012)

[KuSiKa12] Kumar, L., Singh, H., Kaur, R.: Web Analytics and Metrics: A Survey. International conference on advances in computing, communications and informatics (ICACCI'12), 2012

[LarVod10] Larman, C., Vodde, B.: Practices for Scaling Lean & Agile Development. Addison-Wesley, Upper Saddle River (2010)

[Lawley07] Lawley, B.: Expert Product Management: Advanced Techniques, Tips and Strategies for Product Management & Product Marketing. Happy About, Cupertino, CA (2007)

[Lazzar04] Lazzaro, N.: Why We Play Games: Four Keys to More Emotion Without Story. Report, XEO Design (2004)

[LeCBölPe13] Le Callet, P., Böller, S., Perkis, A., et al: Qualinet White Paper on Definitions of Quality of Experience. European network on quality of experience in multimedia systems and services, 2013

[Leffing16] Leffingwell, D.: SAFe® 4.0 Reference Guide: Scaled Agile Framework® for Lean Software and Systems Engineering. Addison-Wesley, Upper Saddle River (2016)

[Leffing11] Leffingwell, D.: Agile Software Requirements. Addison-Wesley, Upper Saddle River (2011)

[LeKaVä07] Lehtola, L., Kauppinen, M., Vähäniitty, J.: Strengthening the Link from Business Decisions to Requirements Engineering: Long-term Product Planning in Software Product Companies. 15th IEEE international requirements engineering conference (RE'07), New Delhi, India, 2007

[LickKitt16] Lick, P., Kittlaus, H.-B.: Software Product Categories in the Automotive Industry and How to Manage Them. International conference on software business (ICSOB 2016), Ljubljana, Slovenia, 2016

[LinWuWen12] Lin, C., Wen, Z., Tong, H., Griffiths-Fisher, V., Shi, L., Lubensky, D.: Social network analysis in enterprise. Proc. IEEE. **100**(9), 2759–2776 (2012)

[LinFen03] Linden, A., Fenn, J.: Understanding Gartner's Hype Cycles. Gartner Group, Mai (2003)

[LoKaVäKo09] Loehtola, L., Kauppinen, M., Vähäniitty, J., Komssi, M.: Linking business and requirements engineering: is solution planning a missing activity in software product companies? Requir. Eng. **14**(2), 113–128 (2009)

[LuDaWeBr16] Lucassen, G., Dalpiaz, F., van der Werf, J., Brinkkemper, S.: Improving agile requirements: the quality user story framework and tool. Requir. Eng. **21**(3), 383–403 (2016)

[LynnAkg01] Lynn, G.S., Akgün, A.E.: Project visioning: its components and impact on new product success. J. Prod. Innov. Manag. **18**(6), 374–388 (2001)

[LyAbVaWr99] Lynn, G.S., Abel, K.D., Valentine, W.S., Wright, R.C.: Key factors in increasing speed to market and improving new product success rates. Ind. Mark. Manag. **28**(4), 319–326 (1999)

[MaglFri14] Maglyas, A., Fricker, S.: Preliminary Results from the Software Product Management State-of-Practice Survey. International conference on software business (ICSOB 2014), Paphos, Cyprus, 2014

[MagNiSmo13] Maglyas, A., Nikula, U., Smolander, K.: What are the roles of software product managers? An empirical investigation. J. Syst. Softw. **86**(12), 3071–3090 (2013)

[MagNiSmo12] Maglyas, A., Nikula, U., Smolander, K.: What Do Practitioners Mean When They Talk About Product Management? 20th IEEE international requirements engineering conference (RE), Chicago, IL, USA, 2012

MagNiSmoFri17 Maglyas, A., Nikula, U., Smolander, K., Fricker, S.: Core software product management activities. Journal of Advanced in Management Research. **14**(1), 23–45 (2017)

[MaiGiRob04] Maiden, N., Gizikis, A., Robertson, S.: Provoking creativity: imagine what your requirements could be like. IEEE Softw. **21**(5), 68–75 (2004)

[MatMat98] Matheson, D., Matheson, J.: The smart organization: creating value through smart R&D. Harvard Business School Press, Cambridge (1998)

[MathKou10] Mathioudakis, M., Koudas, N.: Twittermonitor: Trend Detection over the Twitter Stream. 2010 ACM SIGMOD international conference on management of data, 2010

[Maurya12] Maurya, A.: Running Lean: Iterate from Plan a to a Plan that Works. O'Reilly and Associates, Sebastopol, CA (2012)

[McFarland12] McFarland, C.: Experiment!: Website Conversion Rate Optimization with A/B and Multivariate Testing. New Riders Publishing, Berkeley (2012)

[McGee04] McGee, K.: Heads Up—How to Anticipate Business Surprises and Seize Opportunities First. Harvard Business School Press, Boston (2004)

[McGiTuLa78] McGill, R., Tukey, J., Larsen, W.: Variations of box plots. Am Stat. **32**(1), 12–16 (1978)

[McGrat01] McGrath, M.E.: Product Strategy for High Technology Companies, 2nd edn. McGrawHill, New York (2001)

[MehlBiSt14] Mehler-Bicher, A., Steiger, L.: Augmented Reality – Theorie und Praxis, 2. überarbeitete Auflage. DeGruyter, München (2014)

[MessSzy03] Messerschmitt, D.G., Szyperski, C.: Software Ecosystem—Understanding an Indispensable Technology and Industry. MIT Press, Cambridge (2003)

[Meyer08] Meyer, R.: Partnering with SAP. Books on Demand, Norderstedt (2008)

[MileHub94] Miles, M., Huberman, A.: Qualitative Data Analysis. Sage Publications, Thousand Oaks, CA (1994)

[MilWed13] Miller, P., Wedell-Wedellsborg, T.: Innovation as Usual. How to Help Your People Bring Great Ideas to Life. Harvard Business Review Press, Boston (2013)

[MilMor99] Miller, W.L., Morris, L.: 4th Generation R&D: Managing Knowledge, Technology, and Innovation. Wiley, New York (1999)

[Mintzb13] Mintzberg, H.: The Rise and Fall of Strategic Planning: Reconceiving Roles for Planning, Plans, Planners. Free Press, New York (2013)

[MinAhlLa08] Mintzberg, H., Ahlstrand, B., Lampel, J.: Strategy Safari: The Complete Guide Through the Wilds of Strategic Management, 2nd edn. Financial Times Prentice Hall, Harlow (2008)

[MivBen14] Mival, I., Benyon, D.: User experience (UX) design for medical personnel and patients. In: Fricker, S., Thümmler, C., Gavras, A. (eds.) Requirements Engineering for Digital Health. Springer, Cham (2014)

[Moore93] Moore, J.F.: Predators and prey: a new ecology of competition. Harv. Bus. Rev. **71**(3), 75–83 (1993)

[Moore04] Moore, G.A.: Inside the Tornado: Strategies for Developing, Leveraging, and Surviving Hypergrowth Markets. Harper, New York (2004)

[Moore04a] Moore, G.A.: Darwin and the demon: innovating with established enterprises. Harv. Bus. Rev. **82**(7–8), 86–92 (2004)

[Moore08] Moore, G.A.: Dealing with Darwin: How Great Companies Innovate at Every Phase of Their Evolution. Portfolio/Penguin, New York (2008)

[Moore14] Moore, G.A.: Crossing the Chasm. Harper, New York (2014)

[Myers00] Myers, J.H.: Measuring Customer Satisfaction: Hot Buttons and Other Measurement Issues. American Marketing Association, Chicago (2000)

[NagHogZ14] Nagle, T.T., Hogan, J.E., Zale, J.: The Strategy and Tactics of Pricing—A Guide to Growing More Profitably, 5th edn. Pearson, Harlow (2014)

[NagHog05] Nagle, T.T., Hogan, J.E.: What Is Strategic Pricing? Strategic Pricing Group Insights (Monitor Group) (2005)

[NeeSiCe98] Neef, D., Siesfeld, G.A., Cefola, J. (eds.): The Economic Impact of Knowledge. Elsevier, Amsterdam (1998)

[OshKoWil11] Oshri, I., Kotlarsky, J., Willcocks, L.P.: The Handbook of Global Outsourcing and Offshoring, revised 2nd edn. Palgrave Macmillan, London (2011)

[OstPign14] Osterwalder, A., Pigneur, Y.: Value Proposition Design. Wiley, Hoboken, NJ (2014)

[OstPign10] Osterwalder, A., Pigneur, Y.: Business Model Generation. Wiley, New York (2010)

[PearEnsl04] Pearce, C.L., Ensley, M.D.: A reciprocal and longitudinal investigation of the innovation process: the central role of shared vision in product and process innovation teams (PPITs). J. Organ. Behav. **25**(2), 259–278 (2004)

[Peine14] Peine, K.: Situative Gestaltung des IT-Produktmanagements, Unv. Diss., Universität Stuttgart (2014)

[PepRyl06] Peppard, J., Rylander, A.: From value chain to value network: insights for mobile operators. Eur. Manag. J. **24**(2–3), 128–141 (2006)

[PetWoh10] Petersen, K., Wohlin, C.: Measuring the flow in lean software development. Softw. Pract. Exp. **41**(9), 975–996 (2010)

[PhaFarPr07] Phaal, R., Farrukh, C., Probert, D.: Strategic roadmapping: a workshop-based approach for identifying and exploring strategic issues and opportunities. Eng. Manag. J. **19**(1), 3–12 (2007)

[PhaFarPr04] Phaal, R., Farrukh, C., Probert, D.: Technology roadmapping—a planning framework for evolution and revolution. Technol. Forecast. Soc. Chang. **71** (1), 5–26 (2004)

[Pichler10] Pichler, R.: Agile Product Management with Scrum. Addison-Wesley, Upper Saddle River (2010)

[PoBoeLi05] Pohl, K., Böckle, G., van der Linen, F.: Software Product Line Engineering. Springer Science & Business Media, Heidelberg (2005)

[PohlBoLi05] Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, Berlin (2005)

[PohlRupp11] Pohl, K., Rupp, K.: Requirements Engineering Fundamentals. Rocky Nook Computing, Santa Barbara (2011)

[Popp15] Popp, K.M.: Best Practices for Commercial Use of Open Source Software. Books on Demand, Norderstedt (2015)

[Popp13] Popp, K.M.: Mergers and Acquisitions in the Software Industry—Foundations of Due Diligence. Books on Demand, Norderstedt (2013)

[Popp12] Popp, K.M.: Leveraging Open Source Licenses and Open Source Communities in Hybrid Commercial Open Source Business Models. International workshop on software ecosystems (IWSECO 2012), Cambridge, MA, USA, 2012

[Popp11a] Popp, K.M.: Software industry business models. IEEE Softw. **28**(4), 26–30 (2011)

[Popp11b] Popp, K.M.: Hybrid Revenue Models of Software Companies and Their Relationship to Hybrid Business Models. International workshop on software ecosystems (IWSECO 2011), Brussels, Belgium, 2011

[PoppMey10] Popp, K.M., Meyer, R.: Profit from Software Ecosystems. Books on Demand, Norderstedt (2010)

[Popp10] Popp, K.M.: Goals of Software Vendors for Partner Ecosystems—A Practitioner's View. International conference on software business (ICSOB 2010), Jyväskylä, Finland, 2010

[PortHepp15] Porter, M.E., Heppelman, J.: How smart, connected products are transforming companies. Harv. Bus. Rev. **93**(10), 96–114 (2015)

[PortHepp14] Porter, M.E., Heppelman, J.: How smart, connected products are transforming competition. Harv. Bus. Rev. **92**(11), 64–88 (2014)

[Porter08] Porter, M.E.: The five competitive forces that shape strategy. Harv. Bus. Rev. **86**(1), 78–93 (2008)

[Porter98] Porter, M.E.: Competitive Strategy. Free Press, New York (1998)

[Porter85] Porter, M.E.: Competitive Advantage. Free Press, New York (1985)
[Porter79] Porter, M.E.: How competitive forces shape strategy. Harv. Bus. Rev. **57**, 137–145 (1979)
[PragMark16] Pragmatic Marketing, Inc.: Pragmatic Marketing Framework (2016)
[Pritch15] Pritchard, C.L.: Risk Management: Concepts and Guidance, 5th edn. CRC Press, Boca Raton (2015)
[PruAdl06] Pruitt, J., Adlin, T.: The Persona Lifecycle: Keeping People in Mind Throughout Product Design. Elsevier, Amsterdam (2006)
[PruGru03] Pruitt, J., Grudin, J.: Personas: Practice and Theory. ACM conference on designing for user experiences, San Francisco, CA, USA, 2003
[Rahim15] Rahim, M.A.: Managing Conflict in Organizations, 4th edn. Transaction Publishers, Lewiston (2015)
[RaiRicMe07] Raiffa, H., Richardson, J., Metcalfe, D.: Negotiation Analysis—The Science and Art of Collaborative Decision Making. The Belknap Press of Harvard University Press, Cambridge (2007)
[RajlBenn00] Rajlich, V., Bennett, K.: A staged model for the software life cycle. IEEE Comput. **33**(7), 66–71 (2000)
[RaaMoBia13] Raatikainen, M., Komssi, M., Dal Bianco, V.: Industrial Experiences of Organizing a Hackathon to Assess a Device-centric Cloud Ecosystem. IEEE 37th annual computer software and applications conference (COMPSAC 2013), Kyoto, Japan, 2013
[Reichh96] Reichheld, F.F.: The Loyalty Effect: The Hidden Force Behind Growth, Profits, and Lasting Value. Harvard Business School Press, Cambridge (1996)
[RegnBSO08] Regnell, B., Berntsson Svensson, R., Olsson, T.: Supporting roadmapping of quality requirements. IEEE Softw. **25**(2), 42–47 (2008)
[RegnBrin05] Regnell, B., Brinkkemper, S.: Market-driven requirements engineering for software products. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements. Springer, Berlin (2005)
[RegHNBH01] Regnell, B., Höst, M., Natt och Dag, J., Beremark, P., Hjelm, T.: An industrial case study on distributed prioritization in market-driven requirements engineering for packaged software. Requir. Eng. **6**(1), 51–62 (2001)
[Ries11] Ries, E.: The Lean Startup. Crown Business, New York (2011)
[RobRob06] Robertson, S., Robertson, J.: Mastering the Requirements Process, 2nd edn. Addison-Wesley, Upper Saddle River (2006)
[Royce70] Royce, W.: Managing the Development of Large Software Systems. IEEE WESCON (1970)
[Rubin12] Rubin, K.S.: Essential Scrum: A Practical Guide to the Most Popular Agile Process. Addison-Wesley, Upper Saddle River (2012)
[RufEbe04] Ruffin, M., Ebert, C.: Using open source software in product development: a primer. IEEE Softw. **21**(1), 82–86 (2004)
[Ruhe10] Ruhe, G.: Product Release Planning: Methods, Tools and Applications. CRC Press, Boca Raton (2010)
[RuheSal05] Ruhe, G., Saliu, M.: The art and science of software release planning. IEEE Softw. **22**(6), 47–53 (2005)
[RusKan03] Rust, R.T., Kannon, P.K.: E-service: a new paradigm for business in the electronic environment. Commun. ACM. **46**(6), 37–42 (2003)
[Samuel03] Samuelson, P.: Trade secrets vs. free speech. Commun. ACM. **46**(6), 19–23 (2003)
[Sawhney16] Sawhney, M.: Putting products into services. Harv. Bus. Rev. **94**, 82–89 (2016)
[Schien02] Schienmann, B.: Kontinuierliches Anforderungsmanagement. Addison-Wesley, Upper Saddle River (2002)
[Schmid03] Schmid, K.: Lösungen für Probleme des Requirements Engineering für Produktlinien. Softwaretechnik Trends. **23**(1), 20–21 (2003)

[Schmidt02] Schmidt, M.: The Business Case Guide. Solution Matrix, Boston (2002)
[SchoHeTB07] Schobbens, P., Heymans, P., Trigaux, J., Bontemps, Y.: Generic semantics of feature diagrams. Comput. Netw. **51**(2), 456–479 (2007)
[SchKuPop13] Schütz, S., Kude, T., Popp, K.M.: The Impact of Software-as-a-Service on Partner Management. International conference (ICSOB 2013), Potsdam, Germany, 2013
[Schwaber02] Schwaber, K.: Agile Software Development with Scrum. Pearson International, New York (2002)
[Schwind07] Schwind, M.: Dynamic Pricing and Automated Resource Allocation for Complex Information Services. Springer, Berlin (2007)
[SheeGall15] Sheen, R., Gallo, A.: HBR Guide to Building Your Business Case. Harvard Business Review Press, Boston (2015)
[SegMeDu11] Segetlija, Z., Mesarić, J., Dujak, D.: Importance of Distribution Channels—Marketing Channels—for National Economy. 22nd CROMAR congress: marketing challenges in new economy, Pula, Croatia, 2011
[ShPlCoJa13] Shneiderman, B., Plaisant, C., Cohen, M., Jacobs, S.: Designing the User Interface: Strategies for Effective Human-Computer Interaction, 5th revised edn. Addison-Wesley, Upper Saddle River (2013)
[ShoWar07] Shore, J., Warden, S.: The Art of Agile Development. O'Reilly Media, Sebastopol, CA (2007)
[SodhSodh07] Sodhi, M.N., Sodhi, N.N.: Six Sigma Pricing: Improving Pricing Operations to Increase Profit. Pearson FT Press, Upper Saddle River (2007)
[SolisWang11] Solis, C., Wang, X.: A Study of the Characteristics of Behaviour Driven Development. IEEE 37th EUROMICRO conference on software engineering and advanced applications, Oulu, Finland, 2011
[SongMon98] Song, M., Montoya-Weiss, M.: Critical development activities for really new versus incremental products. J. Prod. Innov. Manag. **15**(2), 124–135 (1998)
[Standish16] Standish Group: Chaos Report (2016)
[SuthSchw13] Sutherland, J., Schwaber, J.: The scrum guide. http://www.scrumguides.org (2013)
[SvGoFTSM10] Svahnberg, M., Gorschek, T., Feldt, R., Torkar, R., Saleem, S., Mu, S.: A systematic review on strategic release planning models. Inf. Softw. Technol. **52** (3), 237–248 (2010)
[TapWil06] Tapscott, D., Williams, A.D.: Wikinomics—How Mass Collaboration Changes Everything. Portfolio/Penguin, New York (2006)
[Teece10] Teece, D.J.: Business models, business strategy and innovation. Long Range Plann. **43**(2), 172–194 (2010)
[Tessarol07] Tessarolo, P.: Is integration enough for fast product development? An empirical investigation of the contextual effects of product vision. J. Prod. Innov. Manag. **24**(1), 69–82 (2007)
[TetGar15] Tetlock, P.E., Gardner, D.: Superforecasting—The Art and Science of Prediction. Crown, New York (2015)
[Thomp14] Thompson, L.: The Mind and Heart of the Negotiator, 6th edn. Pearson Prentice Hall, Upper Saddle River (2014)
[Trendow13] Trendowicz, A.: Software Cost Estimation, Benchmarking, and Risk Assessment: The Software Decision-Makers' Guide to Predictable Software Development. Springer, Berlin (2013)
[Tukey58] Tukey, J.W.: The teaching of concrete mathematics. Am Math Mon. **65**(1), 1–9 (1958)
[UlrEpp11] Ulrich, K., Eppinger, S.: Product Design and Development. McGrawHill, New York (2011)
[VähRau05] Vähäniitty, J., Rautiainen, K.: Towards an Approach for Managing the Development Portfolio in Small Product-Oriented Software Companies. 38th annual Hawaii international conference on system sciences (HICSS'05), Hawaii, USA, 2005

[VAKaPoJa13] van Angeren, J., Kabbedijk, J., Popp, K.M., Jansen, S.: Managing software ecosystems through partnering. In: Jansen, S., Brinkkemper, S., Cusumano, M.A. (eds.) Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry, pp. 85–102. Edward Elgar Publishing, Cheltenham (2013)

[Vasudeva14] Vasudeva, V.N.: Open Source Software and Intellectual Property Rights. Klüwer Information Law Series, Alphen aan den Rijn (2014)

[Vogels14] Vogels, W.: The story of Apollo—Amazon's deployment engine. http://www.allthingsdistributed.com/2014/11/apollo-amazon-deployment-engine.html (2014). Accessed 25 Dec 2016

[Waldo08] Waldo, J.: Scaling in games and virtual worlds. Commun. ACM. **51**(8), 38–44 (2008)

[Waltl13] Waltl, J.: Intellectual Property Modularity in Software Products and Software Platform Ecosystems. Books on Demand, Norderstedt (2013)

[WBNVB06] van de Weerd, I., Brinkkemper, S., Nieuwenhuis, R., Versendaal, J.M., Bijlsma, A.: On the Creation of a Reference Framework for Software Product Management: Validation and Tool Support. International workshop on software product management (IWSPM'06), Minneapolis, MN, USA, 2006

[WMDUHW05] Weill, P., Malone, T.W., D'Urso, V.T., Herman, G., Woerner, S.: Do Some Business Models Perform Better than Others? A Study of the 1000 Largest US Firms. Working Paper No. 226, MIT Center for Coordination Science, Boston (2005)

[Weinst04] Weinstein, A.: Handbook of Market Segmentation: Strategic Targeting for Business and Technology Firms, 3rd edn. Haworth Press, Philadelphia (2004)

[WeiHug05] Weiss, J., Hughes, J.: Want collaboration? Accept—and actively manage—conflict. Harv. Bus. Rev. **83**(3), 93–101 (2005)

[WenSny00] Wenger, E., Snyder, W.: Communities of practice: the organizational frontier. Harv. Bus. Rev. **78**(1), 139–146 (2000)

[Wiegers03] Wiegers, K.: Software Requirements. Microsoft Press, Redmond (2003)

[WohAur05] Wohlin, C., Aurum, A.: What is Important when Deciding to Include a Software Requirement in a Project or Release? International symposium on empirical software engineering (ISESE 2005), Noosa Heads, QLD, Australia, 2005

[XuJYHKO09] Xu, Q., Jiao, R., Yang, X., Helander, M., Khalid, H., Opperud, A.: An analytical kano model for customer need analysis. Des. Stud. **30**(1), 87–110 (2009)

[ZarRub00] Zartman, I.W., Rubin, J.Z.: The study of power and the practice of negotiation. In: Zartman, I.W., Rubin, J.Z. (eds.) Power and Negotiation, pp. 3–28. University of Michigan Press, Ann Arbor (2000)

[ZowCou05] Zowghi, D., Coulin, C.: Requirements elicitation: a survey of techniques, approaches, and tools. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements. Springer, Berlin (2005)

Index

A
Agile, 1, 2, 38, 39, 46, 52, 121, 129, 134, 152, 154, 157, 160, 163, 166, 169, 171, 173, 174, 179, 204, 207, 223, 224, 227, 228, 262
Availability, 10, 16, 21, 53, 105, 110, 115, 123, 146, 156, 165, 183, 208, 219, 266

B
Brand awareness, 234
Bundling, 20, 88, 90, 91, 94
Business case, 34, 50, 82–83, 86, 91, 141, 151, 165, 205, 263, 264
Business model, 4, 9, 15, 21–33, 37, 40, 52, 63, 72, 86, 87, 89, 100, 112, 179, 191, 203, 212, 246, 249, 256, 258, 263, 264
  archetype, 22
  canvas, 23–25, 50, 51, 61, 65, 72, 118, 179
Business plan, 37, 50, 83, 86, 91, 117, 118, 121, 218
Business-to-business (B2B), 36, 38, 49, 56, 58, 59, 62, 65, 69, 72, 73, 89, 96, 112, 175
Business-to-consumer (B2C), 36, 49, 56, 58, 59, 69

C
Caretaker, 31, 41, 43, 220
Category maturity life cycle, 176, 181
Channel, 50, 62, 66, 71, 73–77, 88, 89, 99, 100, 115, 128, 131, 147, 148, 195, 217, 221, 231, 237, 238, 245
  conflicts, 73–76
  mix, 74, 76
Charging, 87, 88, 92, 95, 103
Cloud computing, 14–16, 258
Commodity, 5, 16–18, 94, 230
Company Board, vi
Compatibility, 18, 37, 38, 67, 68, 71, 100, 131, 162
Competitive analysis, 63, 94, 128, 210, 215
Competitor, 18, 20, 21, 60–64, 80, 88, 94, 96, 99, 106, 122, 135, 141, 153, 159, 177–179, 182, 183, 191, 192, 197, 202, 208–210, 214, 215, 233, 241, 264, 265
Constraint, 67, 68, 132, 133, 152, 163, 164, 171, 172, 181
Continuous delivery, 39, 40, 162, 226
Continuous deployment, 226, 227
Continuous integration, 226
Contract, 10, 11, 24, 28, 31, 69, 78, 80, 81, 88, 93, 95, 96, 98, 101–105, 109, 111, 122, 131, 152, 223, 227, 239, 243, 244, 254, 262, 264
Conversion rates, 89
Copyright, 18, 101, 106, 109
Corporate IT organization, 2, 5, 10, 15, 31–33, 39, 40, 45, 49, 95, 105, 211, 219, 256, 258, 259, 262, 264–265
Corporate strategy, 49, 97, 118, 121, 190–196, 204, 205, 207, 209, 218, 232
Cost per lead, 234
Cost structure, 23, 24, 32, 63, 84–86, 91, 115
Cruise ship, 38, 40, 137, 147, 150, 156, 180
Customer, 2, 10, 41, 50, 58, 120, 192, 220, 256
  experience, 72, 76, 154
  relationship, 14, 54, 59, 67, 74, 77, 93, 144, 159, 186, 238, 265
  requirements, 133, 144, 147, 154, 156, 163, 172, 220, 238, 239, 242, 243
  satisfaction, 41, 79, 113, 149, 162, 173, 205, 216, 217, 246–248
  segment, 23, 61, 63, 65, 72–73, 77, 128, 143, 150, 159, 164
Customization, 12, 14, 18, 67–69, 71, 77, 79, 86, 149, 243, 245

H.-B. Kittlaus, S.A. Fricker, *Software Product Management*,
DOI 10.1007/978-3-642-55140-6

D
Delivery model, 14, 15, 50, 67, 69, 71, 73, 75, 101, 117, 226
DevOps, 40, 121, 179, 226, 247
Differentiation, 13, 17, 26, 38, 61, 64, 71, 77, 94, 106, 122, 131, 192
Discount, 21, 89, 93, 96, 175, 220, 239, 242, 243, 254
Distribution, 2, 17, 20, 36, 74, 98, 100, 109–111, 123, 128, 148, 162, 169, 219–221, 238, 239, 243, 245
Documentation, 17, 28, 51, 100, 102, 116–119, 121, 135, 142, 143, 149, 151–155, 157, 171, 172, 184, 195, 218, 223, 245, 249
Dominator, 76, 97, 195

E
Ecosystem, 29, 31, 36, 50, 60, 69, 74, 76, 78, 96–101, 129, 180, 195, 196, 210, 214, 231, 232, 245, 258, 264
Embedded software, 11, 14, 255, 262–263, 265

F
Forecasting, 66, 83, 84, 91, 130
Framework, 4, 7, 33–40, 44, 50, 51, 54, 112, 116, 118, 119, 123, 143, 181, 184, 186, 189, 207, 219, 222, 223, 230, 238, 244, 260–262
Freemium, 30, 95
Functional requirement, 132, 151, 154, 168, 172, 226
Functional support plans (FSPs), 223

H
Hype cycles, 212

I
Icebreaker, 38, 49, 66, 70, 76, 78, 137, 144, 147, 154, 157, 178, 179
Incentive, 32, 242, 244
Independent Software Vendor (ISV), 97
Influencers, 98, 99
InnoTivum, v, vi
Innovation, 18, 21, 55, 64, 97, 100, 106, 113, 125, 135, 138, 140, 163, 191, 192, 198, 203–205, 207, 219, 258
Innovation management, 195, 196, 202–207, 264
Intellectual property, 18, 22, 28, 101, 105–108, 111
International Software Product Management Association (ISPMA), 3, 4, 7, 35, 50, 51, 118, 158, 187, 189, 219, 255, 259–261, 265–266

K
Kano analysis, 140, 143, 146, 150, 203
Key performance indicators (KPIs), 26, 43, 112, 125, 130, 215, 232, 233
Keystone players, 76, 97

L
Law of increasing returns, 18–20, 230
Lean, 1, 17, 39, 89, 114, 163, 179, 205, 207, 222, 229
Liability, 103
License, 2, 4, 10, 15–17, 20, 21, 23–25, 36, 42, 56, 66, 69, 78, 86–88, 91, 93, 95, 101–104, 107–110, 121, 174, 215, 243, 262
License agreement, 103, 104, 109

M
Magic quadrant, 212
Make-or-buy, 79–82
Market, 8, 50, 59, 120, 137, 189, 220, 256
  analysis, 36, 46, 63, 66, 88, 114, 141, 175, 192, 208–215, 217, 260
  leader, 19–21, 88, 100
  research, 62, 98, 101, 128, 135, 141, 159, 210, 212, 215
  segment, 19, 30, 50, 58–63, 66, 70–73, 75, 76, 94, 113, 137, 138, 166, 175, 197, 209, 211, 229, 233
  segmentation, 150
  share, 16, 18, 19, 21, 29, 42, 62, 65, 72, 81, 88, 90, 94, 96, 128, 173, 197, 199, 209, 211, 214, 230, 243, 258
Marketing, 4, 10, 52, 120, 189, 220, 230, 257
Marketing communication (MarCom), 232–234
Marketing strategy, 89, 232, 233, 235
Matrix organization, 249, 250
Minimum viable product (MVP), 37, 38, 70, 137, 166
Mission statement, 191

N
National language support (NLS), 150
Net Promoter Score, 233, 234

Network effect, 2, 19, 20, 31
Niche player, 76, 97, 195
Non-functional requirements, 142, 145

O
Open source, 11, 28–30, 80, 82, 86, 101, 107, 109–110, 123, 143
Orchestration, 3, 4, 36, 48, 220–222, 224, 235, 236, 239, 245, 249–254, 259, 260, 265
Organization, 1, 3, 4, 7, 40, 49, 120, 190, 219, 255
Original Equipment Manufacturer (OEM), 12, 98
Outsourcing, 28, 31, 32, 37, 80, 98, 256

P
Partner 21, 28, 50, 60, 61, 63, 66, 69, 71, 73, 74, 77, 78, 80, 81, 98, 100, 101, 115, 128, 129, 147, 148, 159, 195, 231, 244, 245
  management, 81, 96, 100, 101, 237
  program, 96, 99–101
Patent, 101, 106–108, 215
Performance management, 46, 112–115, 215, 218
Portability, 54, 131, 145
Portfolio management, 33, 36, 189, 194, 196–202, 215
Positioning, 7, 16, 37, 50, 66, 70–77, 79, 82, 96, 101, 117, 128, 140, 153, 183, 189, 197, 220, 230–232, 239, 243, 246–248, 257, 263, 264
Powerboat, 37, 49, 66, 70, 76, 78, 137, 140, 157, 172, 178, 179
Pre-sales, 171, 243
Pricing, 3, 4, 15, 23, 25, 26, 30–32, 37, 50, 62, 66, 69, 70, 75, 76, 82, 83, 86, 88, 89, 91–96, 115, 209, 221, 242, 254, 258
  level, 9, 89, 94, 239, 243
  model, 95
  policy, 94
  structure, 89, 92–94, 209, 243
Prioritization, 36, 37, 82, 115, 130, 136, 153, 157, 164–171, 260
Process improvement, 144, 186, 187, 221
Process model, 222, 264
Product, 1, 7, 10, 49, 120, 190, 219, 255
  analysis, 36, 114, 189, 198, 202, 215–218, 260
  definition, 11, 50, 62, 66–71, 73, 75, 78, 79, 117, 183, 229
  family, 13, 14, 30, 40, 41, 44, 46, 82, 90, 243
  life cycle, 36, 70, 76, 78, 91, 159, 165, 174, 175, 178, 180, 181, 217, 222
  line, 4, 14, 69, 127, 154, 169, 171, 180
  management, 1, 7, 33, 70, 120, 205, 255
  manager, 1, 7, 70, 121, 205, 219, 255
  marketing, 13, 35, 44, 220, 230–234, 236, 237
  name, 52, 56–58
  planning, 3, 4, 33, 34, 36, 37, 46, 118, 119, 127, 129, 159, 173, 177–183, 260, 264, 265
  platform, 4, 13–14, 41, 42, 49, 96, 107
  portfolio, 71, 118, 161, 191, 195–197, 207
  requirement, 37, 62, 67, 101, 119, 130–157, 172, 180, 183, 218, 229, 231, 260
  roadmap, 33, 121–123, 125, 128, 157, 198, 246
  scenario, 36–40, 130, 136, 157, 178–180, 258
  scope, 37, 38, 50, 62, 67, 118, 136, 180, 229
  strategy, 3, 4, 16, 22, 36, 37, 46, 49–118, 120, 128, 165, 174, 182, 183, 191, 192, 195, 202, 207, 208, 221, 224, 227, 231, 236, 243, 260, 265
  vision, 37, 49, 51–55, 62, 67, 117, 120, 121, 125, 128, 129, 250
Product life cycle management (PLM), 33, 114, 119, 173–181, 264
Product-technology roadmap, Professional service, 5, 12, 25–28, 50, 69, 86, 88, 91, 123, 159, 263
Project requirements, 154, 156, 224

Q
Quality assurance, 135, 155
Quality management, 81, 224, 227
Quality requirement, 67, 131, 132, 142, 145, 154, 168, 172

R
Release, 4, 15, 56, 119, 158, 216, 219
  definition, 34
  planning, 33, 37, 38, 77, 119, 130, 136, 149, 150, 157–173, 180, 182, 218, 221, 226
Reliability, 8, 14, 85, 131, 142, 146, 166, 168, 216, 220
Requirement, 2, 11, 58, 119, 131, 201, 220, 259
  engineering, 37, 38, 68, 120, 122, 130, 131, 134–151, 155–157, 160, 180, 224, 227, 238

Requirement (*cont.*)
management, 2, 32, 33, 41, 45, 62, 82, 119, 135, 149, 153, 156, 157, 185, 227, 260
prioritization, 165, 166
triage, 156, 205
Resource management, 80, 199, 207–208, 219, 244, 245, 249, 260, 264
Return on investment (ROI), 82, 88, 106, 170, 198, 234
Revenue, 1, 16, 17, 19, 20, 23, 76–79, 82, 83, 85–88, 90, 93–95, 99, 100, 102, 108, 113, 153, 173, 174, 176, 177, 180, 183, 197, 198, 202, 203, 215
model, 20, 22, 23, 86–91
stream, 21–23, 25, 94, 115
Risk management, 50, 82, 91, 112–115, 118, 183, 264
Roadmap, 37, 38, 50, 66, 70, 118–130

S
Safety, 56, 111
Sales plan, 189, 231, 239, 243
Scaled Agile Framework (SAFe), 39, 224
Scenario, 3, 7, 26, 37–39, 49, 55, 70, 76, 78, 83, 105, 123, 137, 139, 144, 147, 150, 156, 192, 196, 255, 258, 261–265
Scrum, 37, 38, 52, 222, 224
Security, 16, 19, 36, 40, 54, 111, 127, 131, 146
Service, 2, 7, 50, 77, 121, 210, 221, 255
Service level agreement (SLA), 15, 81, 105
Service provider, 16, 32, 62, 104, 105, 254, 262, 263
Service strategy, 50, 69, 70, 73, 77–79, 117
Software-as-a-Service (SaaS), 101, 216
Software ecosystem, 31, 60, 76, 96, 99, 101, 128
Software-intensive systems, 11, 262–263
Software license management, 4
Software product, 2, 7, 11, 40, 50, 121, 214, 222, 255
Software product family, 13, 14, 30, 41
Software product line, 4
Software product management competence model, 265
Software product management (SPM), 1–5, 14, 16, 18, 27, 32–45, 47, 116, 121, 181–185, 189, 214, 218, 219, 230
Software product manager, 1, 2, 7, 16, 21, 33, 36, 38, 40
Software value map, 82, 205
Software vendor, 5, 10, 11, 13, 23–25, 27, 30–33, 36, 40, 44, 49, 59, 68, 69, 73, 77, 86, 96, 98, 99
Solution, 12, 53, 128, 205, 240, 259
Sourcing, 16, 50, 79–82, 117
Specification, 133, 135, 136, 151, 152, 155, 157, 172, 245
Speedboat, 37, 40, 140, 143, 154, 157, 172, 180
Stakeholder, 50, 120, 131, 141, 147, 153, 155, 162, 164, 218, 219, 260
Strategic marketing, 232
Strategic pricing pyramid, 92, 94
Subscription, 25, 94–95, 215
Support, 4, 10, 50, 130, 196, 221, 243, 257
System integrator (SI), 31, 98, 99

T
Target market, 26, 37, 50, 56–58, 62, 66, 70, 73–75, 117, 120, 122, 137, 162, 180
Terms and conditions, 76, 80, 81, 100, 102, 104, 148, 239, 243
Trade secret, 18, 106
Trademark, 57, 106

U
Usability, 62, 131, 142, 155, 168, 216, 228
User, 10, 58, 131, 197, 223, 260
User experience (UX), 19, 58, 67, 71, 127, 203, 205, 223, 224, 228, 229, 260
User experience design (UX design), 67, 71, 223, 224, 228, 229, 260

V
Value proposition, 24, 52, 53, 57, 61, 62, 64, 65, 71–73, 95, 115, 178, 201, 262
Value-added resellers (VARs), 31, 148
Versions, 18
Virtual teams, 249
Vision, 8, 37, 40, 49, 51–55, 62, 67, 117, 120, 121, 124, 125, 128, 129, 139, 153, 172, 176, 191, 196, 205, 210, 212, 215, 250, 260

W
Warranty, 102, 103, 151