Brian Fitzgerald · Klaas-Jan Stol · Sten Minör
Henrik Cosmo

scaling a software business

The Digitalization Journey

Springer Open

www.dbooks.org

Scaling a Software Business

Brian Fitzgerald • Klaas-Jan Stol • Sten Minör • Henrik Cosmo

Scaling a Software Business

The Digitalization Journey

Springer Open

www.dbooks.org

Brian Fitzgerald
Lero - Irish Software Research Centre
University of Limerick
Limerick, Ireland

Klaas-Jan Stol
University of Limerick
Limerick, Ireland

Sten Minör
Mobile and Pervasive Comuting Institute (MAPCI)
Lund University
Lund, Sweden

Henrik Cosmo
Mobile and Pervasive Comuting Institute (MAPCI)
Lund University
Lund, Sweden



ISBN 978-3-319-53115-1 ISBN 978-3-319-53116-8 (eBook)
DOI 10.1007/978-3-319-53116-8

Library of Congress Control Number: 2017941052

© The Editor(s) (if applicable) and The Author(s) 2017. This book is an open access publication
Open Access This book is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.
The images or other third party material in this book are included in the book's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the book's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

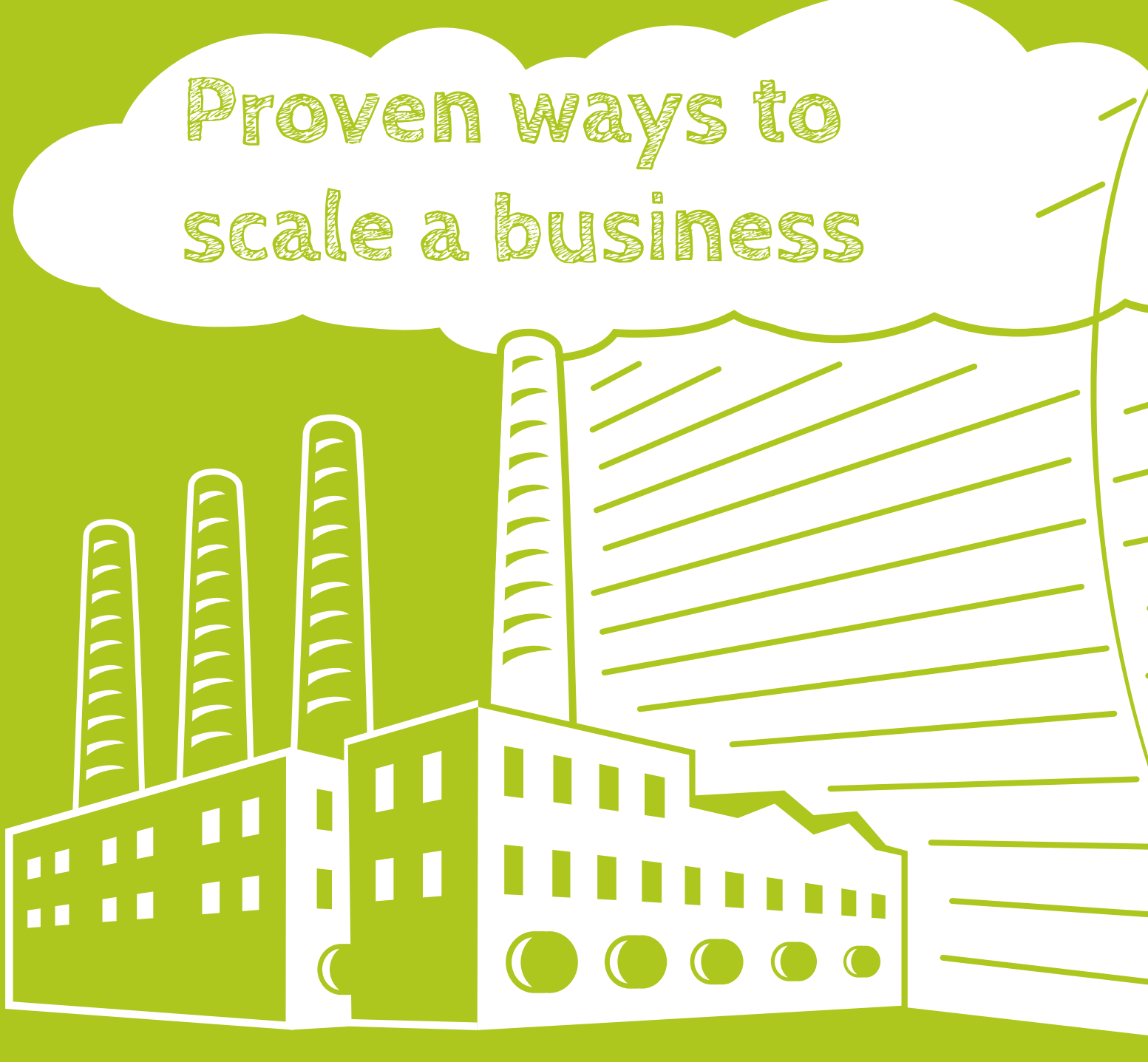
This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Illustrations and design: Ove Jansson, www.monpetitstudio.fr

This work was supported, in part, by the Swedish Innovation Agency VINNOVA and Enterprise Ireland grant IR/2013/0021 and Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero—the Irish Software Research Centre (www.lero.ie).

www.dbooks.org

Proven ways to scale a business



This book is a product of a research project that tackled one of the key challenges currently facing the software industry in Europe, and indeed worldwide, namely, how do we transform our organization as software increasingly becomes a critical part of our business? How can we support the digitalization of assets and offerings?

This book presents the Scaling Management Framework (SMF), which is unique in the sense that it supports the above transformation in three domains: 1) products systems & services, 2) organization & business, and 3) methods & processes. These domains are interdependent and are integrated into a single model in the SMF.



INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

The framework was developed in the SCALARE ("SCAling softwARE") project which was a pan-European project funded under the auspices of the ITEA (Information Technology for European Advancement), a program in the EUREKA Cluster program that supports industry-driven research in the area of software-intensive systems & services.

The intensive case-study approach adopted throughout the project makes the framework highly relevant for today's businesses. The project members have drawn on over 300 years of software engineering and senior management experience conducting more than 30 company case studies companies in Germany, Ireland, Spain and Sweden.

The project was supported by Enterprise Ireland, Science Foundation Ireland, and the Swedish innovation agency, Vinnova.







www.dbooks.org

The digitalization transformation in both industry and society has been ongoing for several decades. Companies in the telecom industry were early movers, whereas the automotive industry is currently in the midst of their digitalization journey. Digitalization implies a shift in focus from hardware and products towards software and services and possibly disruptive business models. This is a game-changer for most companies and the race is on right now.

You may be in a situation where you want to take the next step to increase your usage of software and services in your offerings. You need to do something, but you don't know what options you have, nor what actions you need to take.



SCALARE

The SCALARE project* draws on the Scaling Management Framework (SMF) to provide guidance on creating a roadmap for different scaling approaches such as Open Source, Lean & Agile and global software development.

* http://www.scalare.org

The answer to the question of "how to make a roadmap of our transformation?" can be found in knowledge gleaned from previous experiences, and the SMF provides an easy way to find information that is relevant to you. The framework helps you to understand your own situation. It will help pin down what you want to achieve and how to use this knowledge to search for valid solutions. It also provides guidance to help include all personnel in the process, to use their knowledge about the company.

The SCALARE team's objectives

There are many models that can be used to analyze and assess companies and their products. Often they have a grading system to evaluate whether they are good or bad. However, they are often also quite specialized for specific business domains.

Considering the increasing importance of software throughout the entire company, the model must cover all perspectives of a software business, not just the technical aspects.

This resulted in the integration of the three domains, product, process, and organization into a single model.

The SCALARE team found it very difficult, and indeed misguided, to argue there was only one way to accomplish the transformation. Presenting a smorgasbord of possible ways to improve, there would inevitably be arguments for different and additional practices. Instead, the SMF became a mechanism to design a structure where relevant real experiences could be added.

The model does not produce an evaluation grade for activities, but can be used to capture and describe many different situations.

Even though this reduced the model itself and made it simpler, it was still what we desired and needed.

www.dbooks.org

Co-created by industry and academia professionals

This book has been written by the team that created the SMF. We have reduced the SMF to its essence, a format that's easier to comprehend. But it's not a handbook. It's for senior executives who need a proven guide to approach a transformation project. The book also facilitates communication, by establishing a common terminology.



Brian Fitzgerald

Lero/University of Limerick

Chief Scientist with Lero, the Irish Software Research Centre. He holds the Frederick Krehbiel II Professorship in Innovation in Business and Technology at the University of Limerick. His research focuses on innovative software development approaches.



Sten Minör

MAPCI/Lund University

PhD in Computer Science. Industrial experience from Ericsson and Sony Ericsson where he has held several senior positions within software development and strategy. He is innovation director at MAPCI - Mobile and Pervasive Computing Institute at Lund University and CEO of Sigrun Software Institute.



Ove Jansson

Mon Petit Studio

Visual communicator since 2007. Prior to this, MSc in Computer Science and 12 years in software development and management. He visualizes time machines and other good ideas, such as the SMF.



Klaas-Jan Stol

Lero/University of Limerick

Research Fellow with Lero, the Irish Software Research Centre. He holds a PhD in Software Engineering from the University of Limerick. His research focuses contemporary software development approaches, in particular innersourcing, opensourcing, and crowdsourcing

.



Henrik Cosmo

MAPCI/Lund University

MSc and Tech. Lic. in Software Engineering. 25 years of experience from the global wireless industry. Has held senior positions in large multinational companies as well as in small Silicon Valley startups, in organizations developing and operating software.



Martin Höst
Lund University

Professor in Software Engineering at Lund University, Sweden. His main research interests include software quality and empirical software engineering.



Miguel Oltra Rodríguez
Schneider Electric

MSc in Telecommunication Engineering. 15 years of experience in research projects on topics such as software product lines, OSS practices & SOA.



Anders Sixtensson
Softhouse

MSc and Tech. lic. in Electrical and SW Engineering. 30 years as consultant, manager and business developer in the field of business improvement in SW intensive products and organizations.



Ulf Asklund
Lund University

PhD in Computer Science. Experience both from industry and research projects, with focus on how to improve product lifecycle management, configuration management, and product architecture.



Max Sunemark
Addalot

MSc in Electrical Engineering. Over 20 years of experience as SW developer, project and line manager. Has held several senior manager positions at large international companies.



Carl-Eric Mols
Sony Mobile Communications

Has close to 30 years of experience on telecommunication and mobile industry software, the last 8 years holding the position as Head of Open Source.



Ulrika Bergman
Tieto

Customer Executive, Tieto. Holds a MSc in Electrical Engineering. More than 20 years of experience in large IT projects in telecom and the public sector.



Anders Mattsson
Husqvarna

PhD in Software Engineering. He has 27 years of experience in industrial software development and research with focus on SW architecture and model driven development.

In addition to those on the previous page, others have been instrumental in the creation of this book. Those named above helped to formulate the essence of SMF. In turn, they were helped significantly by many individuals across the project who conducted the industry research, all the case studies that SMF builds on. They are named below:

Jonas Ahnlund, Lund University
Nicolas Martin-Vivaldi, Addalot
Even Andre Karlsson, Addalot
Horst Hientz, Kugler Maag
Hans-Jürgen Kugler, Kugler Maag
Krzysztof Wnuk, Lund University
Ola Morin, Softhouse
Johan Kardell, Softhouse
Marcus Degerman, Softhouse
Donal O'Brien, QUMAS
Joanne O'Driscoll, QUMAS
Ryan O'Sullivan, QUMAS
John Burton, Vitalograph
Ger Hartnett, Goshido

www.dbooks.org


It's in the software

We're in a business where most of our product features have to be realized with software. Adding software to a business is a complex enterprise, whether we start from scratch or take our products to the next level. There are as many ways to organize such product evolutions as there are people involved. However, reality leaves us no choice: we have to take the plunge in order to sustain our business. Fortunately, many companies have already embarked on such journeys. Based on numerous case studies with a variety of companies in different domains, we developed the Scaling Management Framework (SMF). The SMF codifies past experiences and offers systematic guidance to assess our starting point as well as our destination. This book is full of stories of companies, and we describe their journeys using the SMF. Ericsson is one such company, and we start telling their story next.

How a dozen software developers became several thousands in a few years – the story of a technical and an organizational boom



Hello world

The big breakthrough for mobile telephony during the nineties is a part of technological history characterized by intense competition between innovative companies. For Ericsson Mobile Communications, it meant a period of explosive expansion of the mobile phone software development department in Lund, Sweden: growing from a handful of developers, to an army of thousands within a few years. While this story is from the days when mobile telephony was made into an everyday tool for billions of people, it's still highly relevant for all of us interested in scaling a business.

The software organization in Ericsson's mobile phones is a great example of an organization that just keeps growing and growing. It started in 1992 when Ericsson introduced their first GSM phone. At that time, only a dozen people worked on the software for mobile phones. Eleven of them were working with the

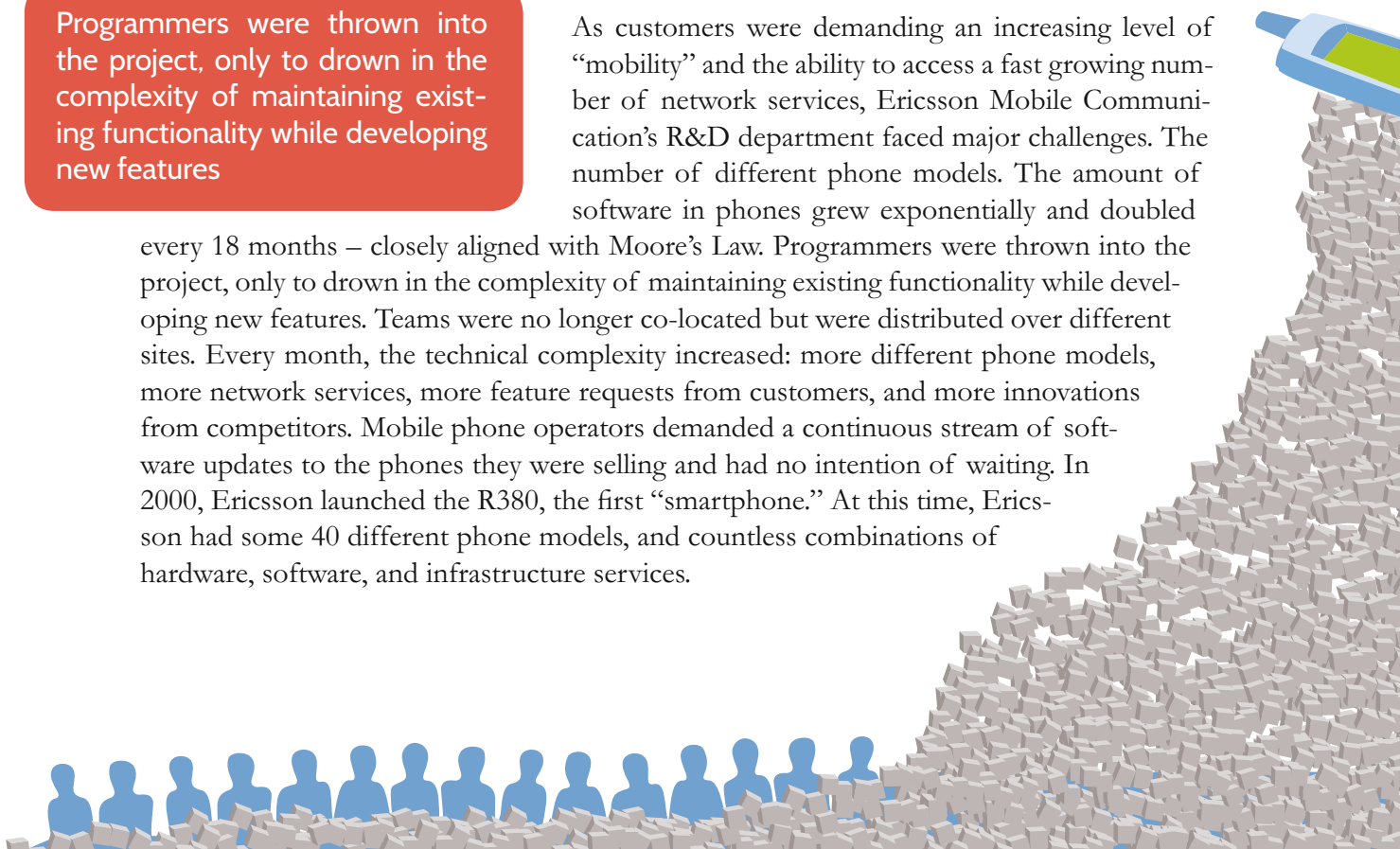
www.dbooks.org

technical and telecoms-related platform software. Only one of the developers was responsible for the phones' user interface. You could use the phone to make calls – but that's about all you could do.

Just five years later, the mobile phone world looked very different. Second generation (2G) GSM offered more powerful services, and operators offered better terms and prices, which led to an increase in mobile phone users. In the latter half of the nineties, email and internet services were no longer exclusively used by academics and companies, but were now available and affordable to the wider public. As technology advanced, customers had growing expectations to have advanced features on all their devices.

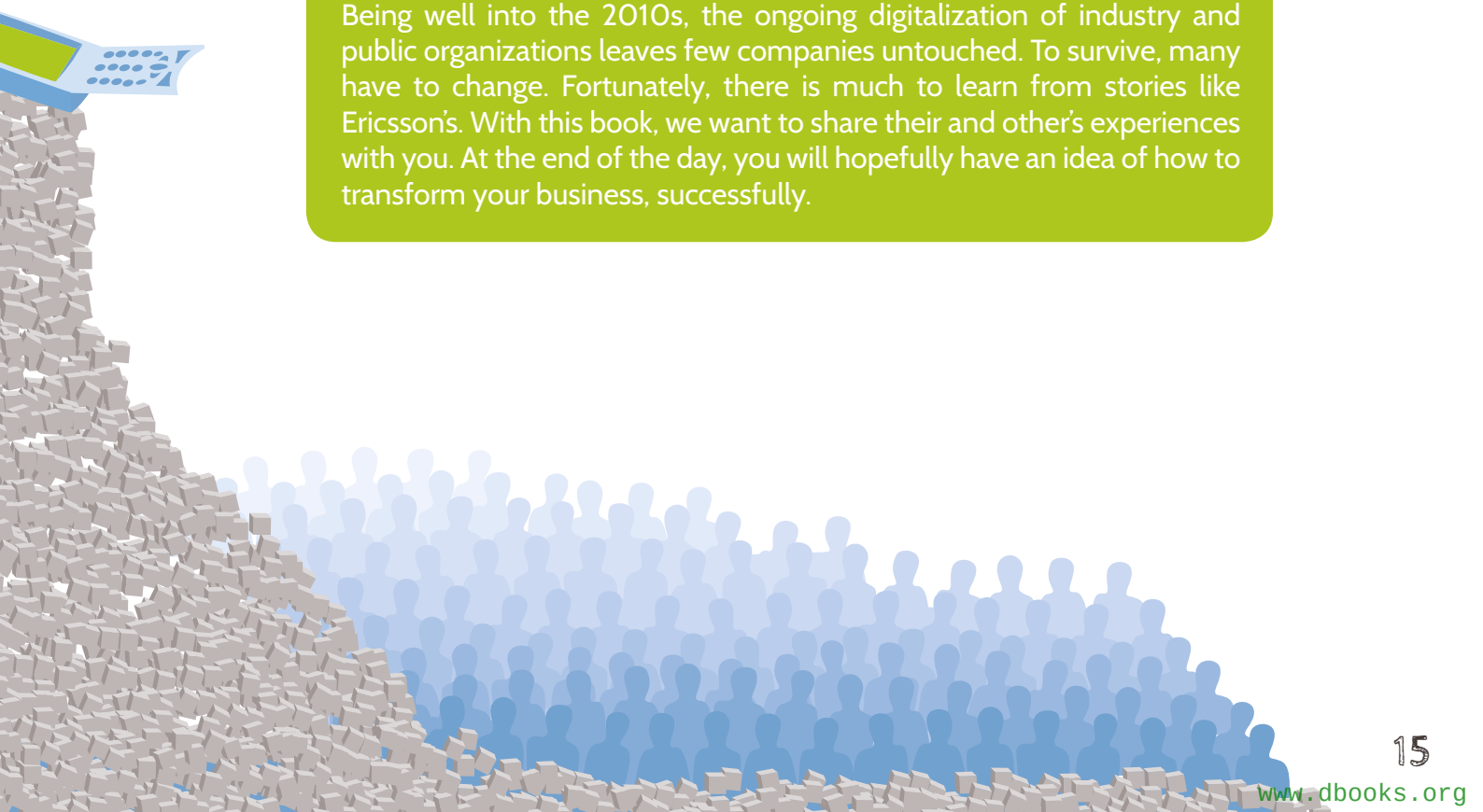
Programmers were thrown into the project, only to drown in the complexity of maintaining existing functionality while developing new features

As customers were demanding an increasing level of "mobility" and the ability to access a fast growing number of network services, Ericsson Mobile Communication's R&D department faced major challenges. The number of different phone models. The amount of software in phones grew exponentially and doubled every 18 months – closely aligned with Moore's Law. Programmers were thrown into the project, only to drown in the complexity of maintaining existing functionality while developing new features. Teams were no longer co-located but were distributed over different sites. Every month, the technical complexity increased: more different phone models, more network services, more feature requests from customers, and more innovations from competitors. Mobile phone operators demanded a continuous stream of software updates to the phones they were selling and had no intention of waiting. In 2000, Ericsson launched the R380, the first "smartphone." At this time, Ericsson had some 40 different phone models, and countless combinations of hardware, software, and infrastructure services.



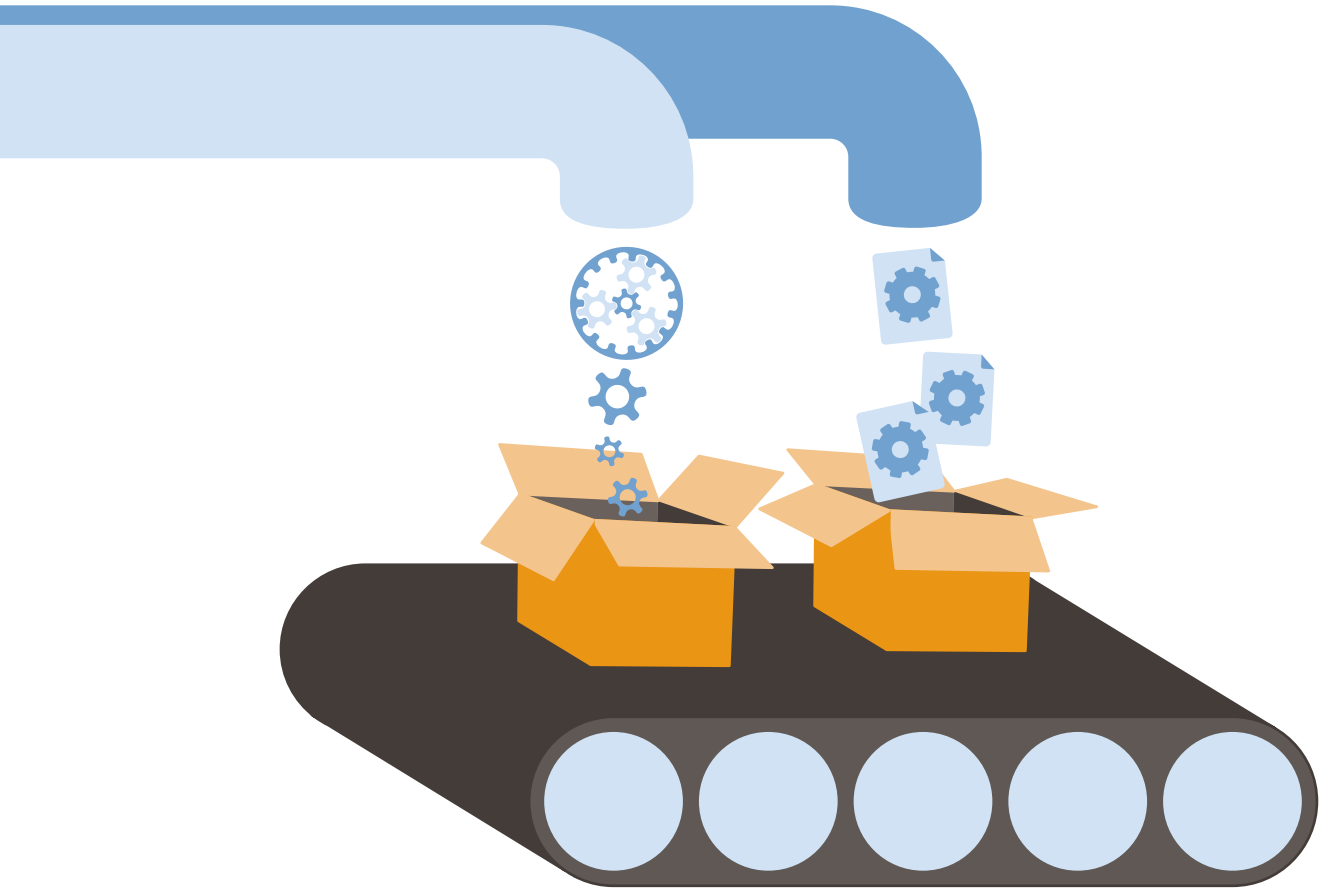
It was time to take the plunge. As the software grew exponentially, the change process had to embrace the entire organization. Product and system architecture, organizational changes, business and development processes – all had to adapt in a coordinated manner towards a common goal. Rules for what can and cannot be done with the software had to be established. The organization needed to think in new ways, to invest in a software architecture that was designed to scale. It became crucial to introduce stringent configuration management to control the wide variety of different software versions.

Being well into the 2010s, the ongoing digitalization of industry and public organizations leaves few companies untouched. To survive, many have to change. Fortunately, there is much to learn from stories like Ericsson's. With this book, we want to share their and other's experiences with you. At the end of the day, you will hopefully have an idea of how to transform your business, successfully.



www.dbooks.org

Common challenges with software



"Our organization has become a software company. The problem is that we haven't realized that yet!" This is how the VP of a major semiconductor manufacturing company, traditionally seen as a classic hardware company, characterized the context in which software solutions were replacing hardware in their products. This organization knew precisely the threshold of reuse level for their hardware components before "designing for reuse" became cost-effective. However, no such sophistication was present in their software processes.

The digitalization of society is changing businesses more rapidly than ever seen before, there are countless other scenarios emerging. It is difficult to find a market domain in which the innovation does not depend on software.

> **Our organization has become a software company.**
> **The problem is that we haven't realized that yet!**

For several years now, we have seen how traditional companies adopt novel and clever business concepts using digital technologies that integrate into our everyday life. Everything that can be digitalized is digitalized, and any data that can be collected is collected.



Technological advances, such as the emergence of cloud-based solutions, mobile devices and social media have paved the way for this evolution. Take smart watches, for example, which are becoming very popular. By adding software to an ordinary watch, it can now access internet services which opens up a wide range of new possibilities and opportunities. The Smart watch is an excellent example of products that have emerged from digitalization.

What motivates these companies to transform their businesses, what are their goals, or to put it differently, what are the business drivers? The primary answer is that software affords development

www.dbooks.org

of new business opportunities. Some companies see radical cost savings from digitalization, where others see revenue growth by creating innovative products and services. So, it's not surprising that traditional industries such as manufacturing are now in the midst of developing their digital strategies. Software has become a critical part of their product offerings, but they need to scale the business systematically in order to not lose momentum, or worse, to lose business altogether.



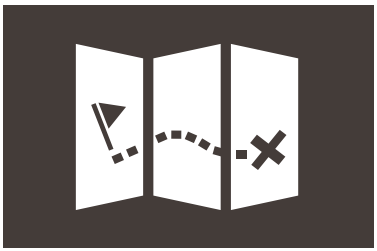
How do they do it? Needless to say, transforming software requires rethinking existing processes as well as incorporating new practices and tools. Studying other companies to see how they approached the transformation is a very useful exercise as it teaches us so many things. All of them have more or less created a new sort of IT organization. In the digitalized company, IT is not only about internal network services and technology, but also deals with business models and the digital platform for customer facing services and products. It's not unlikely that the traditional IT organizations dissolve and merge with development organizations as a consequence.

Obviously, IT is just one of the cogwheels in the digitalized business that need to spin in new ways. The entire organization will need a lift. As this chapter proceeds, it will become clear that it all boils down to a company's motivations to transform: the business drivers.

This is the point of departure for this book. The transformation journey can be done in a variety of ways, but ultimately we can categorize any software transforming activity to one of three domains product, process, or organization.

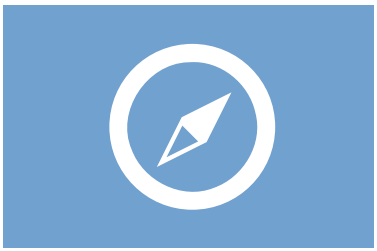
Knowing what needs to be done is quintessential when approaching the digital transformation. This book offers a method that will help your organization in three ways:

1



The map. The SMF canvas is the map of the digital transformation. It helps you in creating a digitalization strategy.

2



The compass. Five groups of common drivers that will help you to find your digital transformation journey.

3



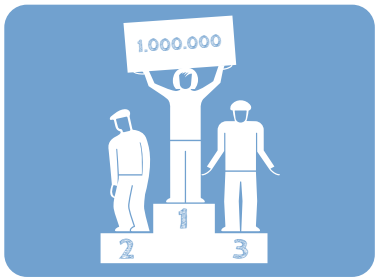
The journeys. An experience database with best practices and lessons learned from past digitalization transformations.

This book embodies three years of research that was conducted in a European project called "SCAling softwARE" – SCALARE for short – which was established by a consortium of partners in Germany, Ireland, Spain, and Sweden.

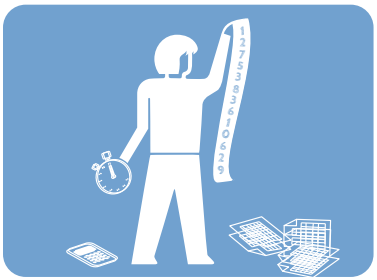
Over 30 case studies were conducted, involving companies in a variety of domains, ranging from pharma to automotive as well as emerging industries that aim to deliver IoT (Internet of Things) products and services.. Each case study is based on in-depth interviews with vice presidents, directors, managers, supervisors and front-line service employees.

www.dbooks.org

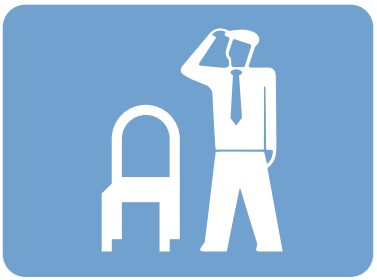
Let's get back to the starting point of this book: why do companies need to transform? Answering this question will help later in this book, when we're pinpointing what journey a company needs to make. To guide you through this book, we have distilled five main reasons to embark on a scaling journey.



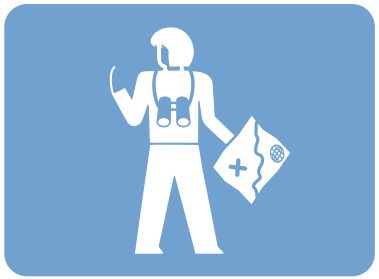
Drive revenue growth and outperform competitors with new business models



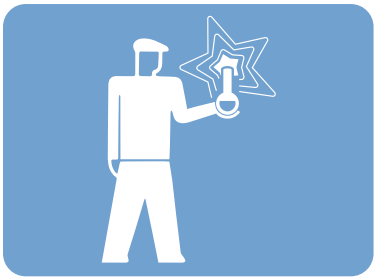
Increase quality, make operational expenditure savings and shorten time-to-market



Deal with organizational challenges (access to qualified personnel, ability to ramp resources, round the clock development, divide the work between different departments, and so on.)



Expand into new markets and geographies



Develop innovative new products and services, innovate in current products and services

There are of course many ways to define and group different business drivers. We have chosen to use the findings from a Harvey Nash CIO Survey conducted in 2016, entitled "Key priorities the board is looking for an IT department to address".

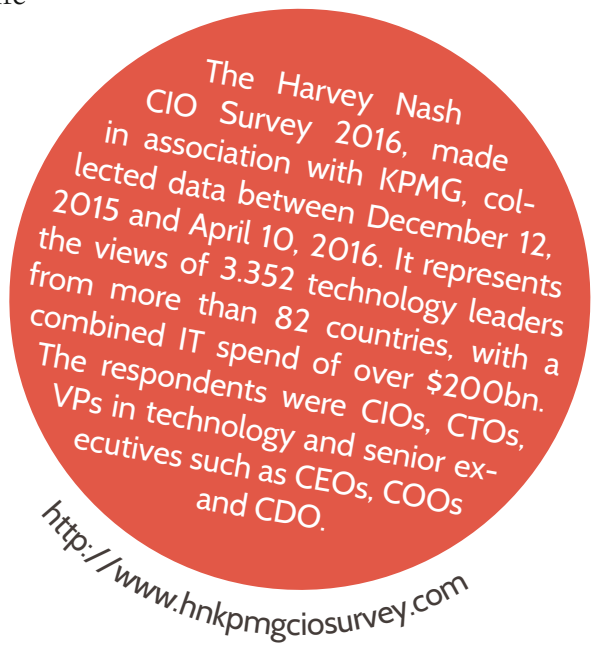
Where does your organization need to scale? Which of the business drivers are key to your organization? You may identify several drivers to match your business plan and digital strategy. You have to bear in mind that this has to be a cross-organizational effort, where the current IT department needs to step up and be part of, or even take the lead. To define the role of the IT department is an essential part of the digital strategy.

Let's get back to the SCALARE project room. The eight standard scenarios, the map and the compass on your scaling journeys, are compilations of the most common repeated patterns that the SCALARE team observed during the various case studies. In other words, a scenario based on real life scaling experiences in companies that all shared similar drivers.

This is how we should approach the scenarios. Which scenarios have similar drivers to ours? Since we may have several drivers to consider, our transformation journey could very well match several scenarios.

Bottom-line:

We have to get a clear picture of what drivers we have.

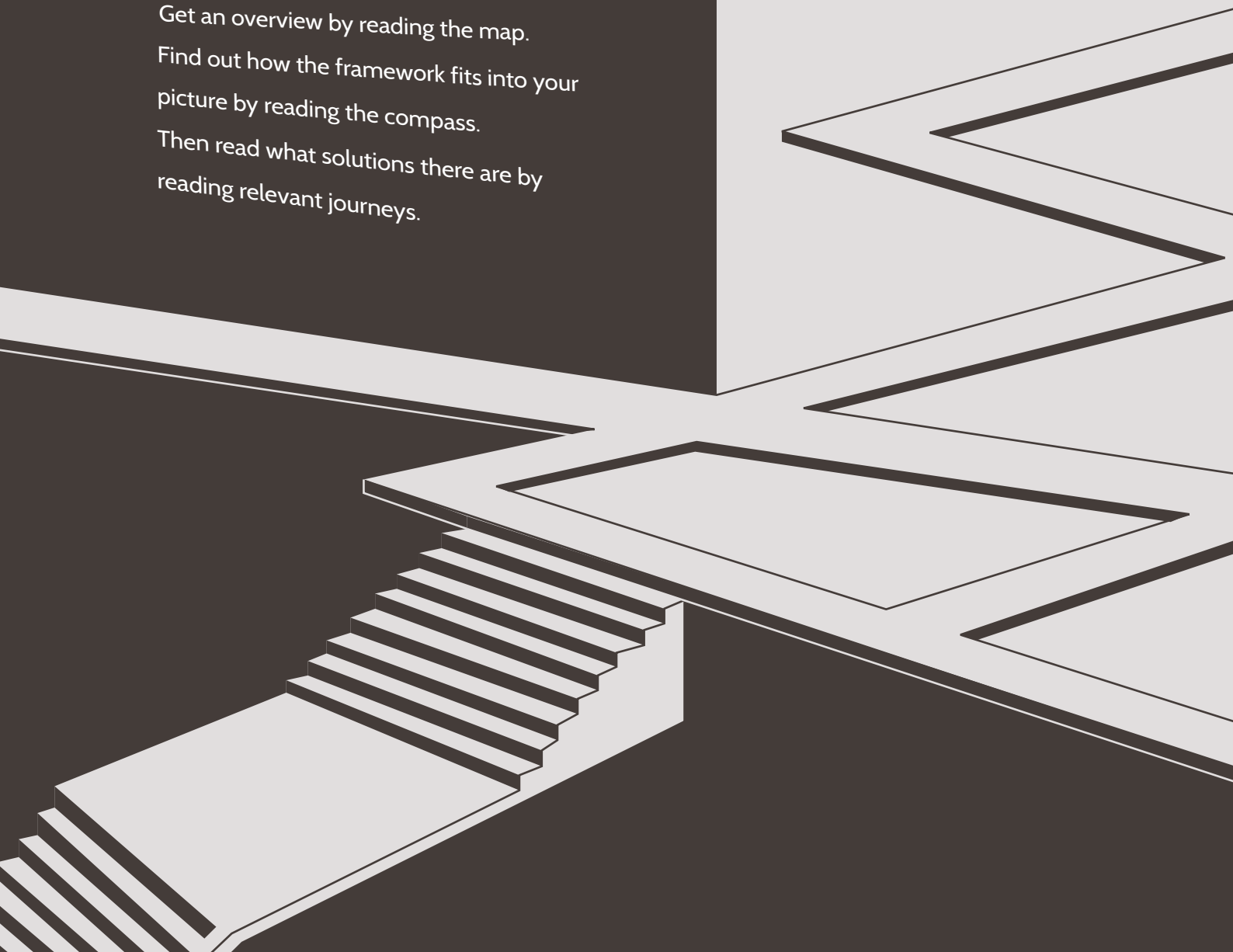


www.dbooks.org

Not to waste time

Get an overview by reading the map.
Find out how the framework fits into your picture by reading the compass.
Then read what solutions there are by reading relevant journeys.

Page



24 The map

The Scaling Management Framework

46 The compass

Ways to find journeys relevant to you

56 The journeys

Travel Brochures and Travel Stories

236 Your journey

How you get there

www.dbooks.org

The

Scaling Managem

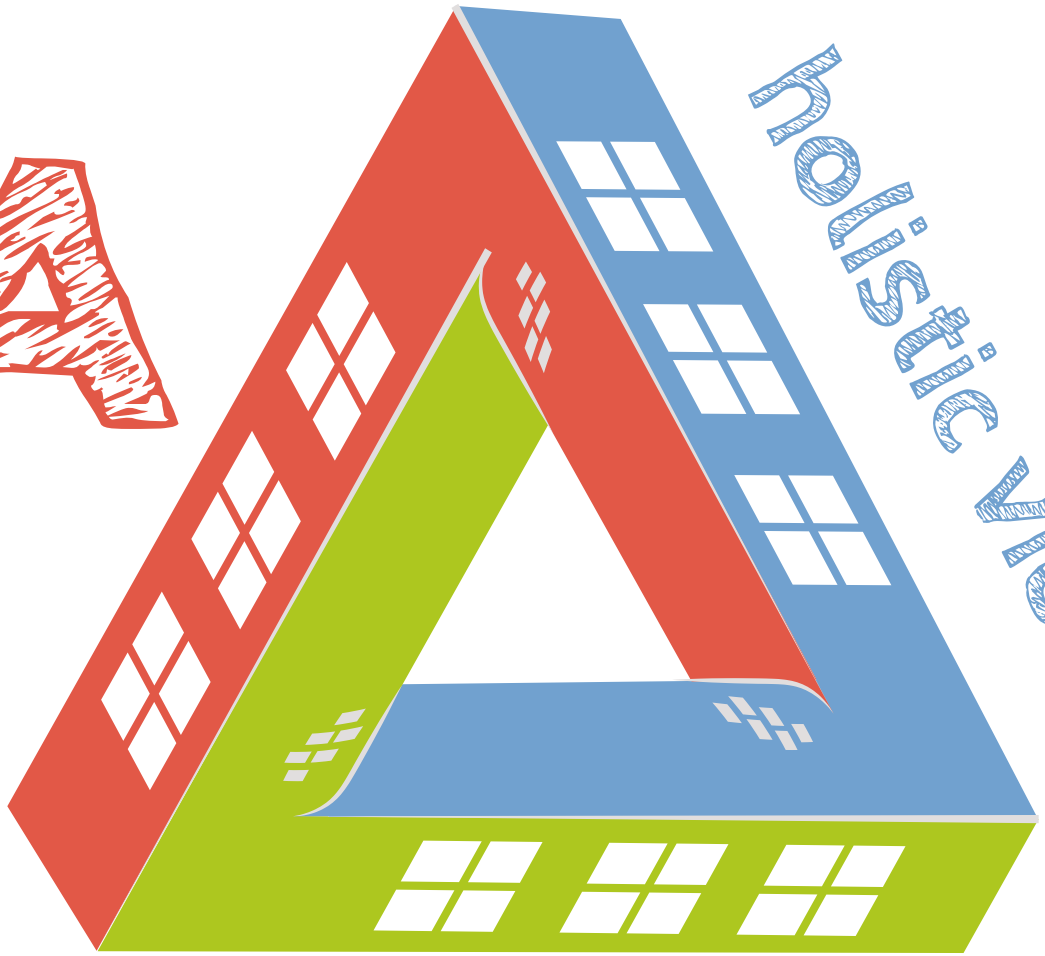
© The Author(s) 2017
B. Fitzgerald et al., *Scaling a Software Business*,
DOI 10.1007/978-3-319-53116-8_1

map

nent Framework

www.dbooks.org

A



holistic view

to change

Scaling software development is a complex enterprise that can be organized in a number of ways. Since the early days of computing, hundreds, if not thousands of software development methods have been proposed. What is becoming increasingly clear, however, is that the software development function affects the whole organization, not only its software developers. For example, as the software development workforce is expanding, the processes that are used may have to be adjusted to facilitate large-scale collaborations. Developing more and larger software-based products requires consideration of architectural strategies such as the adoption of a software product line approach. The SMF covers three scaling domains to capture this complexity: product, organization, and process. It also captures the relationships between these domains.

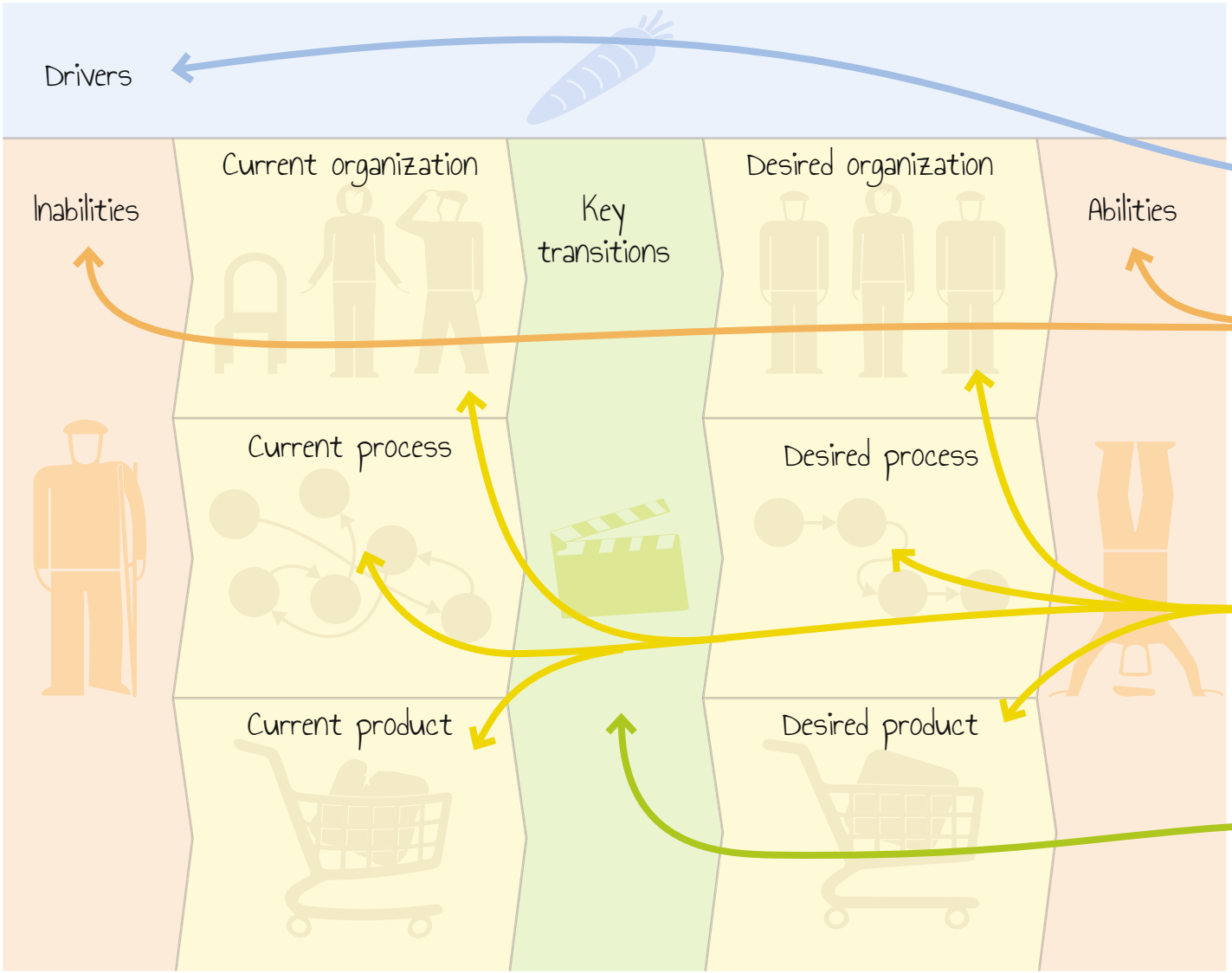
The Scaling Management Framework covers transformations in three domains

To exemplify the scope of the SMF, let's consider the fictive test tool company AutoTest. They have been very successful in their local market, but now have the opportunity to expand to Asia. The SMF may help them in the analysis of their software development and support organization. The SMF is not a tool for analyzing business models, but instead it can be used to analyze scaling changes triggered by for instance the introduction of new products, a specific customer requirement or a new support organization.

The SMF can be used in several different ways to support the digital transformation journey. It offers systematic guidance for decision makers to identify scaling needs, to analyze scaling options and implement transformations in the three scaling domains. Since the SMF clearly defines begin and end states of the transformations, management can easily measure the success of the transformation journey.

The framework is simple yet complete enough to suit all sorts of companies: small as large, local as global. It works equally well for companies that mainly deal with hardware but need to introduce software, as for companies that develop proprietary software and want to engage with Open Source communities.

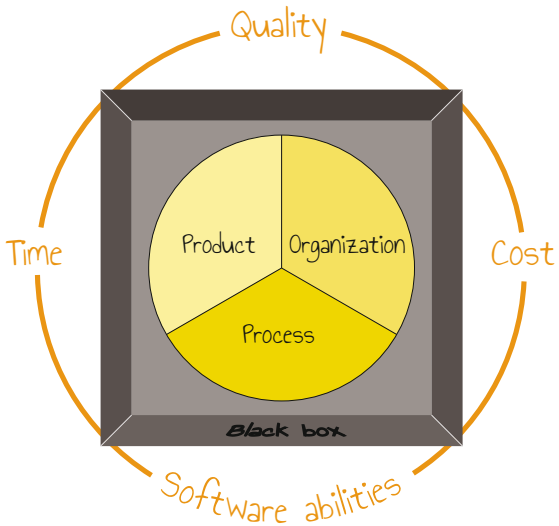
www.dbooks.org


Drivers
Inabilities
Current organization
Key transitions
Desired organization
Abilities
Current process
Desired process
Current product
Desired product

The SMF offers a multi-dimensional view on the software-scaling phenomenon. The SMF canvas, an integral part of the SMF, is a tool for understanding and describing the scaling of software development. It's designed to be visualized on a whiteboard, along with post-it notes. To the left, we have the present, and to the right we have the desired future. The transformation happens in between the two.

It starts with top management to identify the drivers, the reasons to scale. These are typically the outcome of a digitalization strategy.

>> Drivers >>



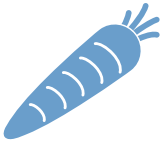
Next we observe the software development as a black box, focusing on performance, quality and other aspects that can be measured without knowing how they work. We know what we spend and how much we generate from the their work. These are basically the complaints and wishes of our top management, matters we also could use to measure the success of the transformation.

Having all this, we're ready to dive into the black box, the software model. Inside we have as well a present to the left and a future to the right. But foremost, it's parted in the three scaling domains organization, process, and product. Our work is to find the root causes to the current inabilities, and for every cause come up with an idea of how we want it to be, ideally. This is done within all three domains and it is important that all inabilities are explained by at least one root cause.

The gap between the present and the future defines the transformation, where the real work is carried out. It's also where we add the real value of SMF, with scenarios that include predefined transitions. In fact, the lot of this book is about scenarios; they are that important. If we know that part of our digitalization for instance includes one of the Open Source transitions, then we'll just add a note about it there, in the key transitions field.

www.dbooks.org

Drivers



The drivers of the need to change.

Inabilities



The inabilities that make the transformation challenging.

Desired abilities



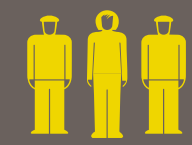
The measurable definition of done.

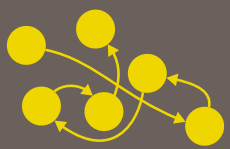
Current set-up

Desired set-up

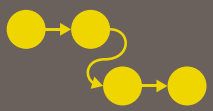


Organization domain





Process domain





Product domain



Transition

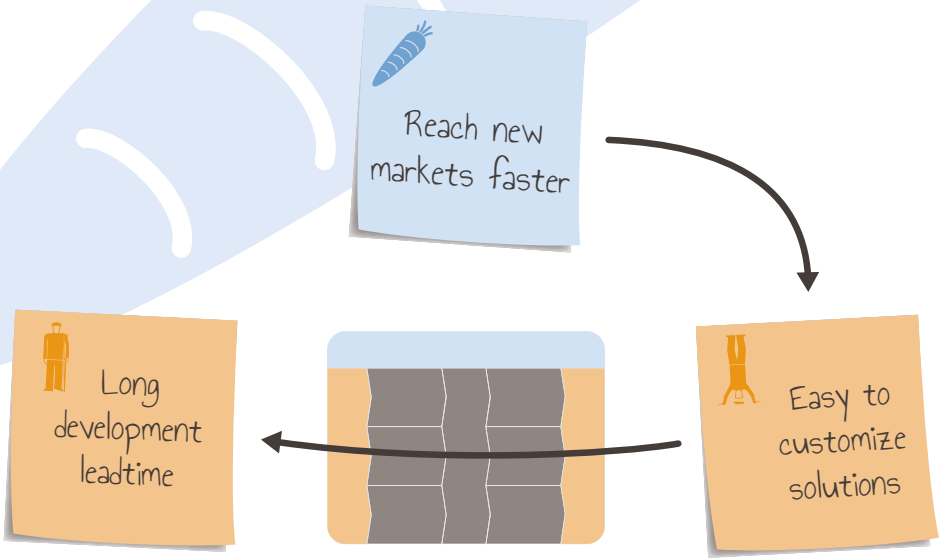


The key activities that are needed to make the transformation possible.

Drivers

Drivers are the external factors that influence the software development. Previous drivers made you build your current product, ways of working and organization. But now, new drivers force you to create new solutions – to transform your software.

We need to clearly formulate why we want to make a change and scale our software development. If these drivers get too vague, also the solutions and eventually the implementation will be unfocused and sometimes never finished. Optimally, the drivers should be based on the company strategy.



www.dbooks.org

It seems simple to create the drivers, but it often turns out to be quite hard to decide on what level the drivers should be defined. For example, is a demand on "a more modularized product" a driver – or instead a possible solution to the driver to "get a more configurable product"? Is actually "a more configurable product" really a driver, or is also this a possible solution to cope with the requirement "to faster reach different markets"? We argue it is the latter and that the previous are solutions to be defined in the domains of the software model.

We might for instance need to pursuit new market opportunities through expansion or a company takeover. Or why not do as Amazon and start selling your internal development platform as a service? Do we need to extend the functionality in our current offering? The mobile phone industry has made that, seeing the mobile phone developed from being a communication device to also be our primary entertainment device. In regulatory requirements we need to comply with safety standards such as ISO 26262 (Road vehicles), or with software maturity standards such as CMMI (Capability Maturity Model Integration). Such requirements will inevitable turn into drivers, since these are tickets to trade.

We provide a short-list of five high-level drivers, and how they can be translated into SMF transitions.

In addition to external drivers, it is also possible to have internal drivers. An internal driver is something you want to achieve, as you believe it will help you move in the right direction even though no external customer or market is requesting it right now. One such example is the company culture. It might not directly affect the close-term business, but will most likely affect the long-term results.

Since drivers are quite diverse and particular to every company, the SMF do not provide any model for analyzing and understanding drivers. Yet they need to be listed and understood before using the rest of the model. In this book, however, we provide a short-list of five high-level drivers, and how they can be translated into SMF transitions.



www.dbooks.org

Software abilities

Drivers represent the trigger for the transformation of a business; they are the reasons to drive a transformation in one of the SMF domains. They are the fuel that starts a journey to get from current software abilities to desired software abilities.

Inabilities or growing pains are what currently stop us from achieving the desired drivers. The inabilities are mostly visible outside of the organization, for instance by other organizations within the company or outside the company by customers. Abilities have to be measurable, in order for us to know if we're getting any better. When we measure the abilities, the organization is considered being a black box.

Most abilities can be put into one of the following categories:

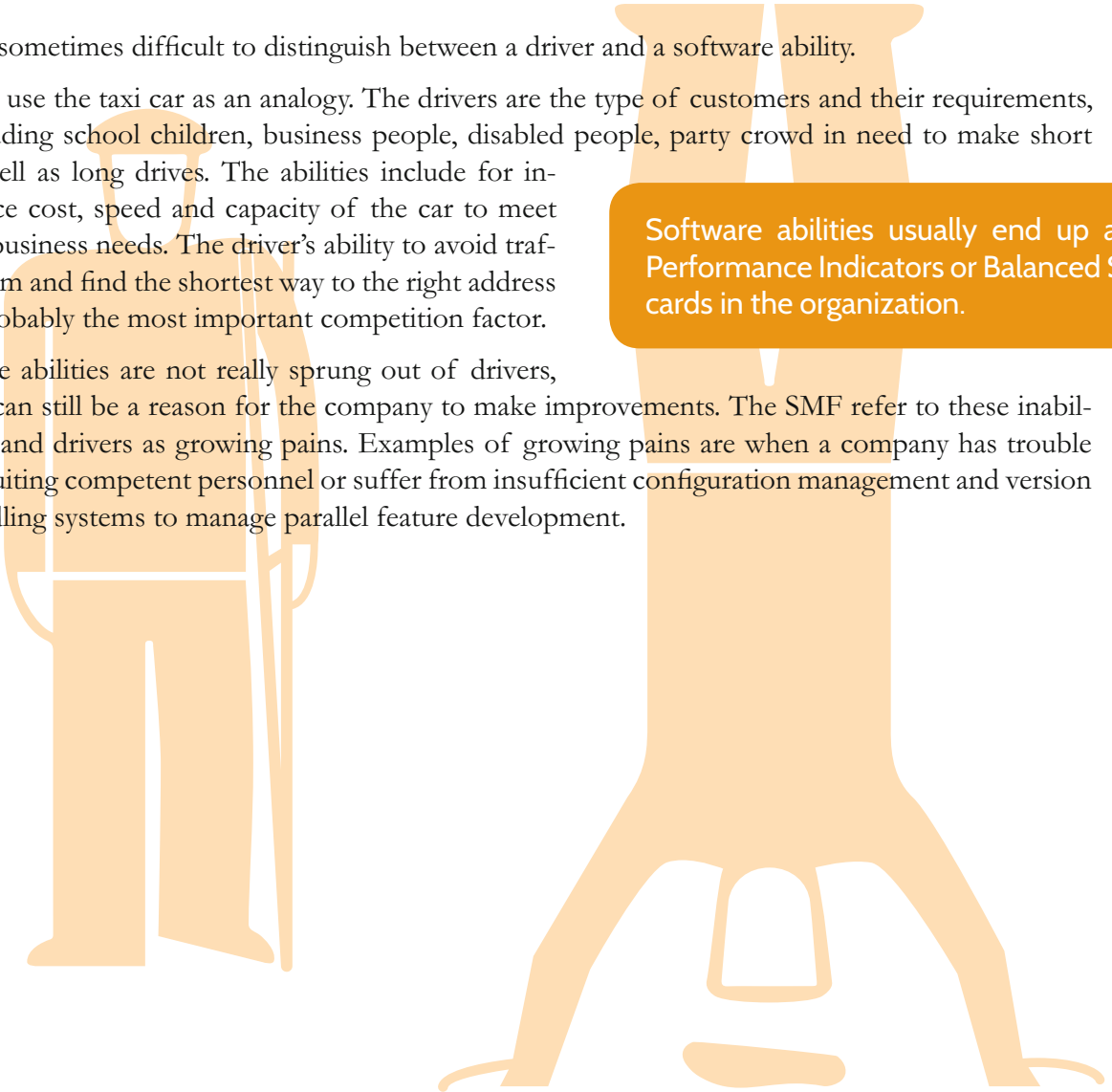
- **Revenue** – How much do we earn from our products or services?
- **Cost** – How much do we invest in development?
- **Speed of development** – How fast is the software being developed? This covers all aspects of speed, from adding or updating a feature to deliver the product or service.
- **Speed of the product or service** – What are the response times for different use cases?
- **Availability of a service**, that is, the amount of time a service is usable.
- **Flexibility** – How fast can we change our development scope? This can be on a small scale like creating a variant, or on a large scale like entering a new market.
- **Quality** – How good is the product or service that we are delivering? Quality includes many aspects, such as safety, security, configurability, compatibility, maintainability, usability, serviceability, and evolvability.

It is sometimes difficult to distinguish between a driver and a software ability.

Let's use the taxi car as an analogy. The drivers are the type of customers and their requirements, including school children, business people, disabled people, party crowd in need to make short as well as long drives. The abilities include for instance cost, speed and capacity of the car to meet the business needs. The driver's ability to avoid traffic jam and find the shortest way to the right address is probably the most important competition factor.

Software abilities usually end up as Key Performance Indicators or Balanced Scorecards in the organization.

Some abilities are not really sprung out of drivers, but can still be a reason for the company to make improvements. The SMF refer to these inabilities and drivers as growing pains. Examples of growing pains are when a company has trouble recruiting competent personnel or suffer from insufficient configuration management and version handling systems to manage parallel feature development.

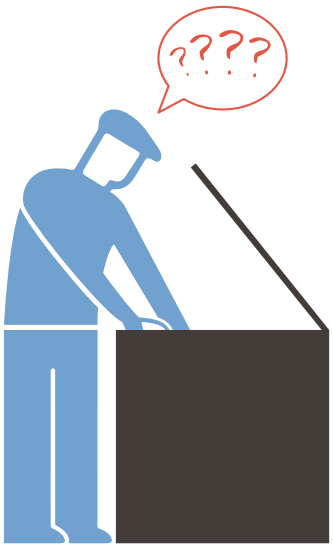


Software model set-up

When we work with the drivers and abilities, we treat software development as a black box. The black box is called the software model.

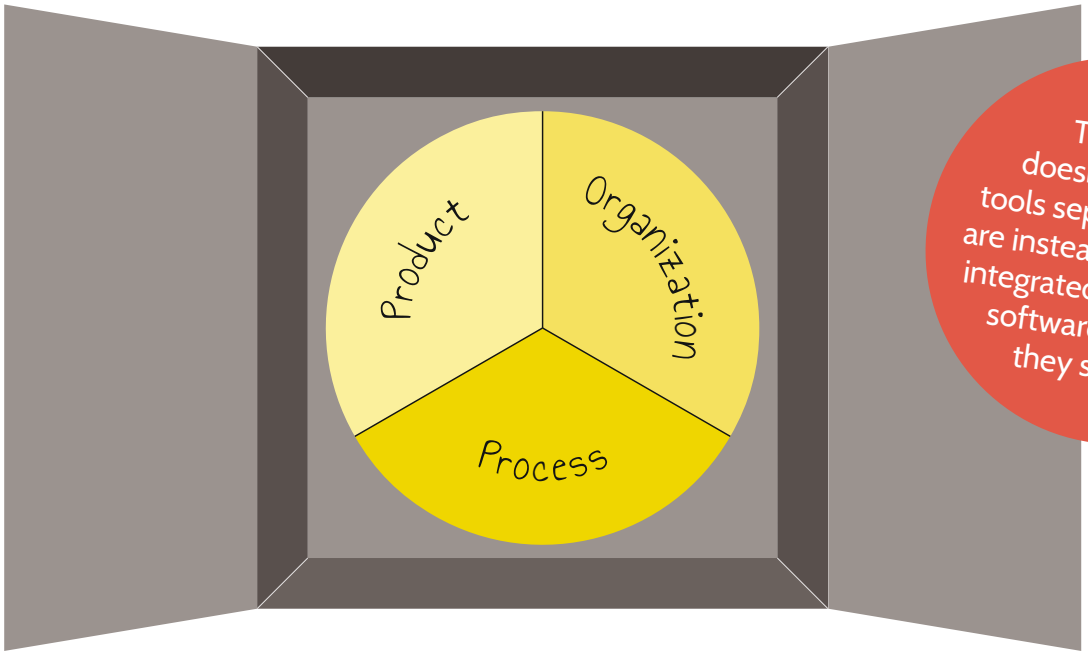
The purpose of the software model domain is to:

- Be able to analyze what we're bad at and what we need to improve in.
- For all improvements, analyze and describe dependencies to make the transformation.



Opening up the box, we see domains as the software organization, how they work, how they are organized and how the product looks. We cannot only focus on one of these domains, but need to look at the entire complex situation. The dependencies to organizations outside the box are usually many and intertwined. And it's not just the present, the future might as well require new relations to be established between the software organization and the rest of the company. For instance, if the company wants to start using Open Source software, the legal organization will be needed.

The canvas is for this reason divided into the three domains: organization, process, and product. A domain covers all aspects that are important to analyze in order to define our important characteristics within the domains. They also define what changes we need to do within the domains to transform and fulfill your drivers.



The SMF doesn't manage tools separately; they are instead regarded as integrated parts of the software domains they support.

Our work is to make an inventory of the current ways we're working, and we should use post-its and document our findings on the canvas. With our drivers as a compass, it's time to start nesting. It is natural to begin with our inabilities. Are there any existing assets we can exploit and are there any roadblocks in the way we're working? It's time to ask all these questions we have on our mind.

Why? Why? Why?

In the following sections we describe the individual domains and their building blocks.

Product domain

The product domain covers the software architecture of your product or service. This is more complex than just describing a software application as a monolith. The offering might be based on services rather than a single product. It might even include a complete ecosystem that defines how products and services can interact and how to configure these.

The SMF divides this domain into three building blocks:

- How the product is structured and managed during development; the creation and managing of the software source code itself. This includes how the software is structured into files and directories, libraries, modules, and interfaces. It also covers the tools that are used in handling of the code such as editors, version handling systems, and issue handling systems. This area is important to achieve reuse of code.
- All mechanisms and tools that are used to produce the executable system from the source code. It includes all types of configuration and parameterization to build the product or service, as well as branching in the version system. It also includes how the system is deployed to the end user. This is typically an important area for continuous deployment and to be able to create customized products.
- How the product is organized when installed and executed by the user. It covers all aspects of interaction with the system when it's in use during operations of the software, including GUI interfaces to the system. Examples of architectures are client/server and cloud technology. This is typically important for efficient execution and to enable incremental updates (new functionality or bug fixes) of installed software.

The product or service is the final result of the whole software development lifecycle effort.

Process domain

The process domain covers all aspects about how the product or service is developed and tested. This domain is divided in two building blocks:

- Engineering covers all activities directly related to producing and testing the product. This includes processes for requirement, architecture, design, coding, integration, and verification. It also includes all tools needed to support modeling, coding, testing and so on. The activities are preferable further divided into the three categories producing, verifying, and correcting.
- Project management covers all activities related to steering, planning and controlling the engineering work, including prioritizing, estimation, risk management, planning, follow up, configuration management, change management, measurements, quality assurance, supplier management, etc. It also includes all tools supporting these activities.

Organization domain

The organization domain covers aspects such as how a company is structured, how the company's culture is and how are each individual employee treated. The organization domain is divided in four building blocks:

- Structure – the organization chart. What sections, departments and teams are there? Who does what in the organization? How is governance implemented? What are the responsibilities and how are decisions made? Also described here is the physical structure of the organization – is there one site or many sites with distributed teams and organizations? Is outsourcing or offshore development in use?
- Culture and leadership describes the general attitude in the organization, how decisions are taken and how changes are perceived and implemented. How is the attitude towards change and improvement? Is the atmosphere OK, are people speaking over the silo borders? Is there a lot of firefighting? Any pro-active work? Is it a learning organization?
- People management describes how the staff is managed. This includes recruitment, performance management, and competence management.
- Improvements describe all activities related to implementing and improving the product architecture, processes, and the organization. This covers as well descriptions, examples, templates, training, measurements, lessons learned, and improvement processes.

www.dbooks.org

Transitions

The key transitions are the most important activities in order to get where we aim; altogether they constitute the very transformation journey, the fun part where all the magic happens.

The transition area of the canvas consists of a set of actions, each one representing a transformation of the software model from a current state to a desired state. Transitions can be so general their descriptions turn into patterns.

Examples of transitions are:

- The organization chooses to outsource; development is made on both sites, but the outsourcing partner makes all test.
- Changing the process from an agile process with morning stand-up meetings and other meetings on-demand, to a waterfall-like process focused on phases in which documents and other artifacts must be ready for hand-over. This requires recurrent meetings to discuss deliverables.
- Increasing efficiency of knowledge transfer. This could be to set up an internal training program for new employees. A transformation such as this might not have any defined current abilities to overcome.

Several cross-domain transitions are usually needed to address all desired abilities (that is, to fulfill the drivers). In the example above, to meet the requirement and lower the development cost, many changes are likely needed. No business situation is the other alike. We have to pay good attention to this phase.

It all fits together

To summarize, the SMF fulfill most companies' need to scale in terms of their existing or non-existing software.

Drivers and abilities gives the structure to describe the reason for scale and the main metrics needed to measure the improvements. Look into case studies with similar drivers and abilities as the business we want to change. Working with drivers and abilities is very important to understand the reason, why you want to scale.

The software model with its three domains and their building blocks gives a structure to describe the actual implementation of the product or service. Each domain studies the building blocks and also the dependencies between implementations in different blocks.

The SMF defines several standard change scenarios, but still most companies have to customize their unique set of changes in order to implement their particular drivers.

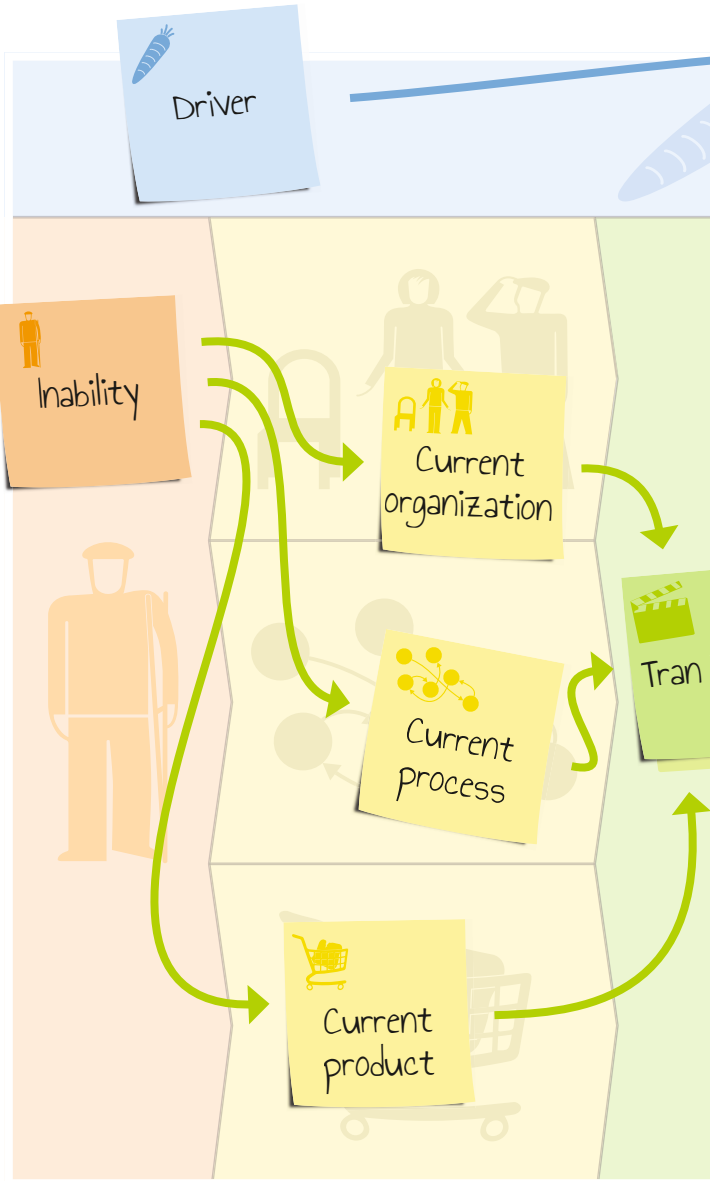
The connection between the domains, how things works in each one of them, and the current inabilities encourage us to think through all domains and their building blocks in order to understand why we have the inabilities. This is part of the self-assessment to understand what needs to be improved.

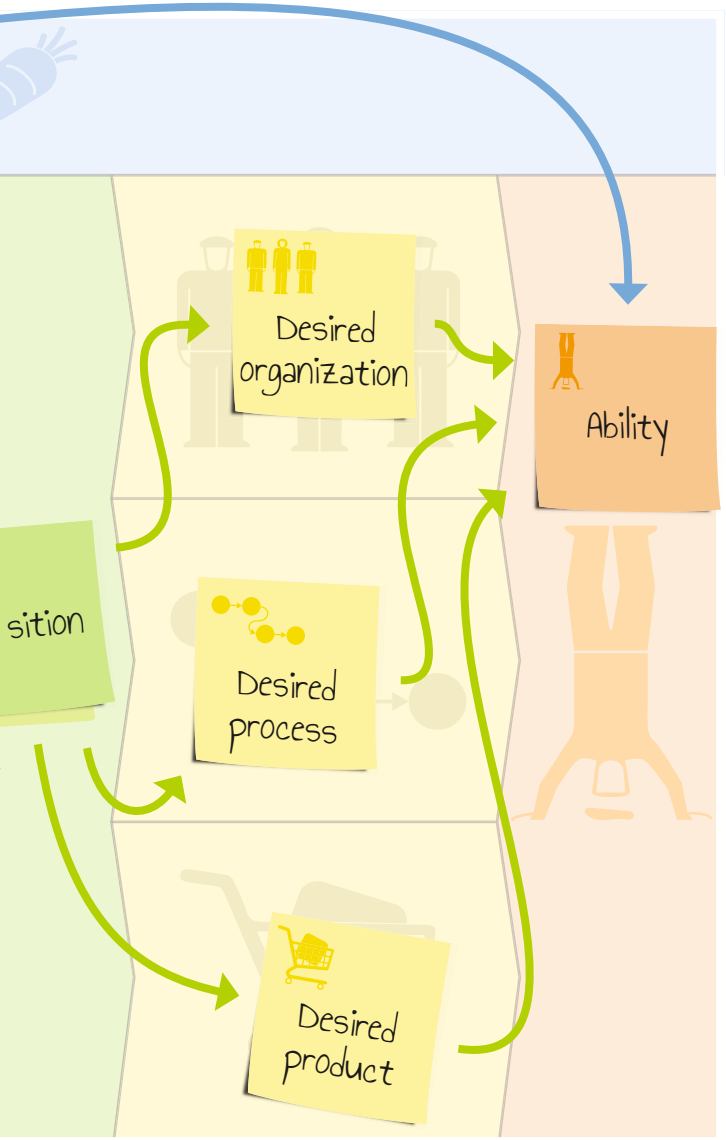
The transitions – capturing the needed change from current situation to desired situation is the most important part of the SMF. Experts within each domain will have to discuss possible changes and their impact. Yes, the SMF defines several standard change scenarios, but still most companies have to customize their unique set of changes in order to implement their particular drivers. Exactly as the SMF was intended to be used.

www.dbooks.org


Why do we do
what we do, as we do?
Why?
Why?
Why?
Why?
Why?
Why?
Why?



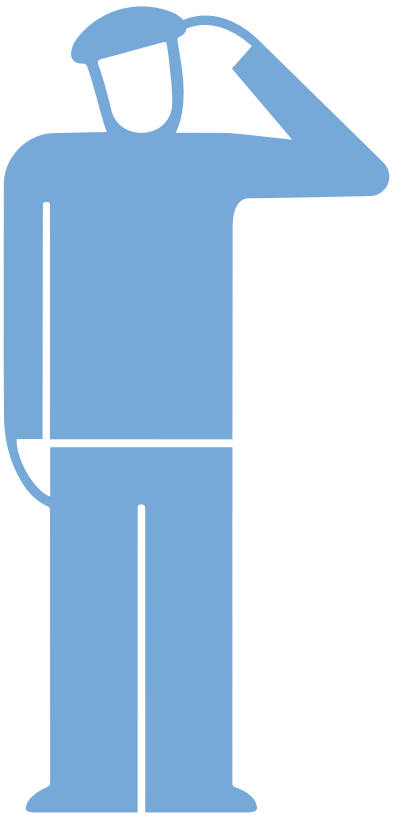

Driver
Inability
Current organization
Current process
Current product
Tran



Why do we want to scale the business?

What abilities would we need to be able to scale successfully?

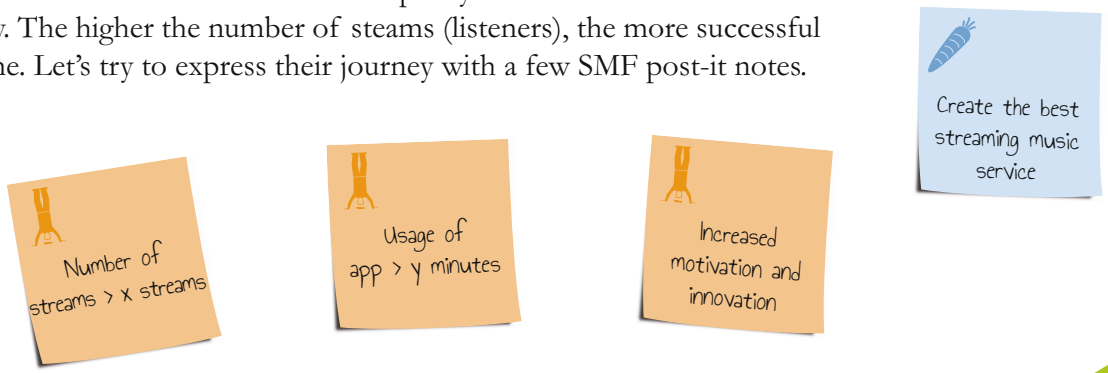
Can we measure the abilities as KPIs?



How the canvas can be used

For this example, we will have a look at Spotify. Its service allows us to search for artists, albums, titles, labels and genres, and access music tracks from major as well as independent labels. Streaming of music has in many regions become the predominant way we listen to music.

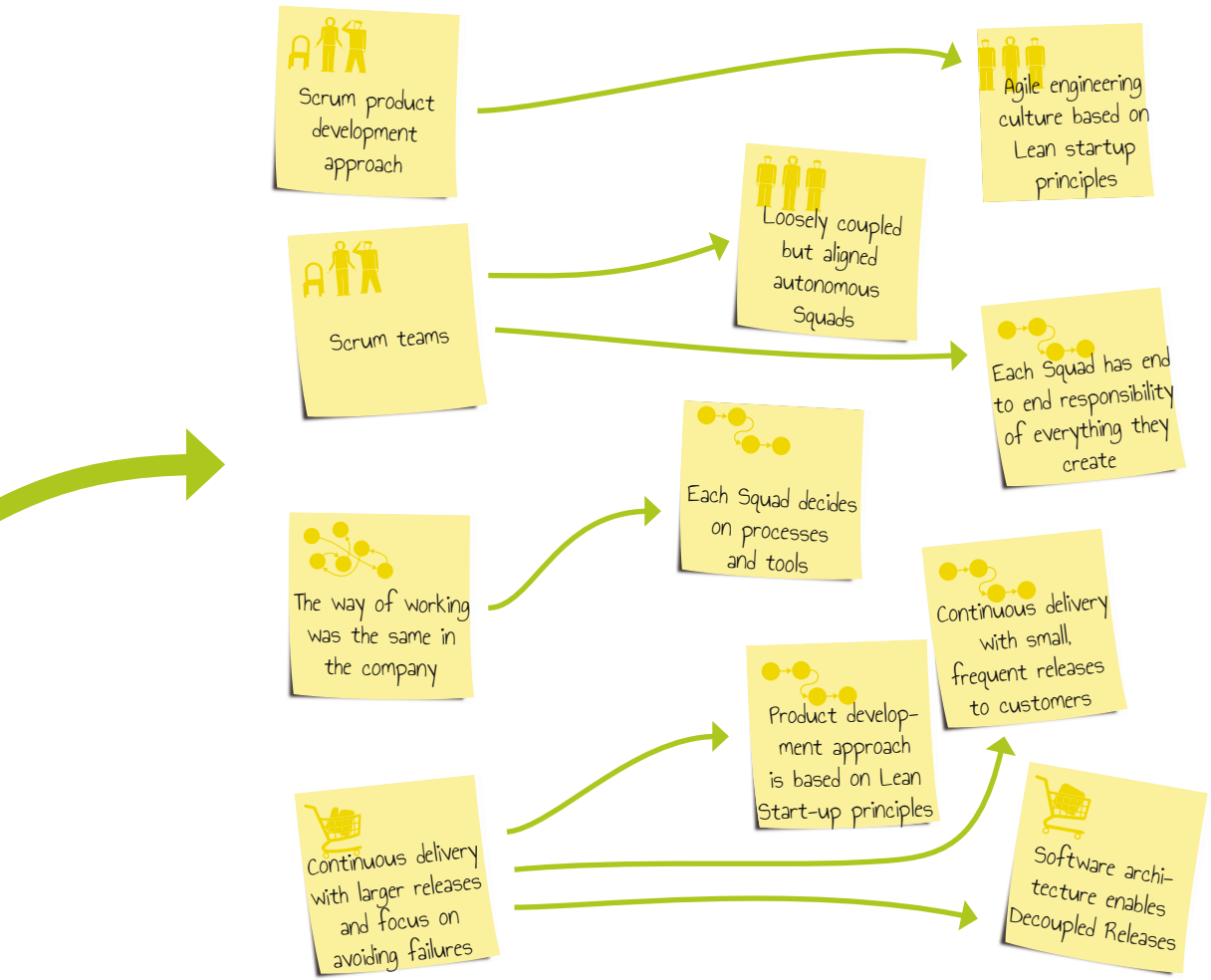
How did Spotify achieve to make the software service so successful? According to the vast amount of articles that have been written on the topic, they simply seem to have decided to "create the best streaming music service". We can assume this was their driver. What they intended to measure was the number of Spotify streams. This was their desired ability. The higher the number of steams (listeners), the more successful they became. Let's try to express their journey with a few SMF post-it notes.



To accomplish what their driver boldly stated, their software engineering department made this transformation.

Most importantly, to stimulate motivation and innovation, they introduced an Agile Engineering Culture. They also organized themselves in loosely coupled but aligned Autonomous Squads (small, self-organized teams). A Squad has end-to-end responsibility of what they build (design, commit, deploy, maintain and operate). They did try the software development methodology SCRUM for a while, but decided quite early to skip this way of working. A more generic agile methodology was simply more relevant for them.

They introduced continuous delivery with small and frequent releases to their customers. The software architecture was changed to enable decoupled releases (synchronized releases were too time-consuming). Their product development approach was based on Lean Start-up* principles.



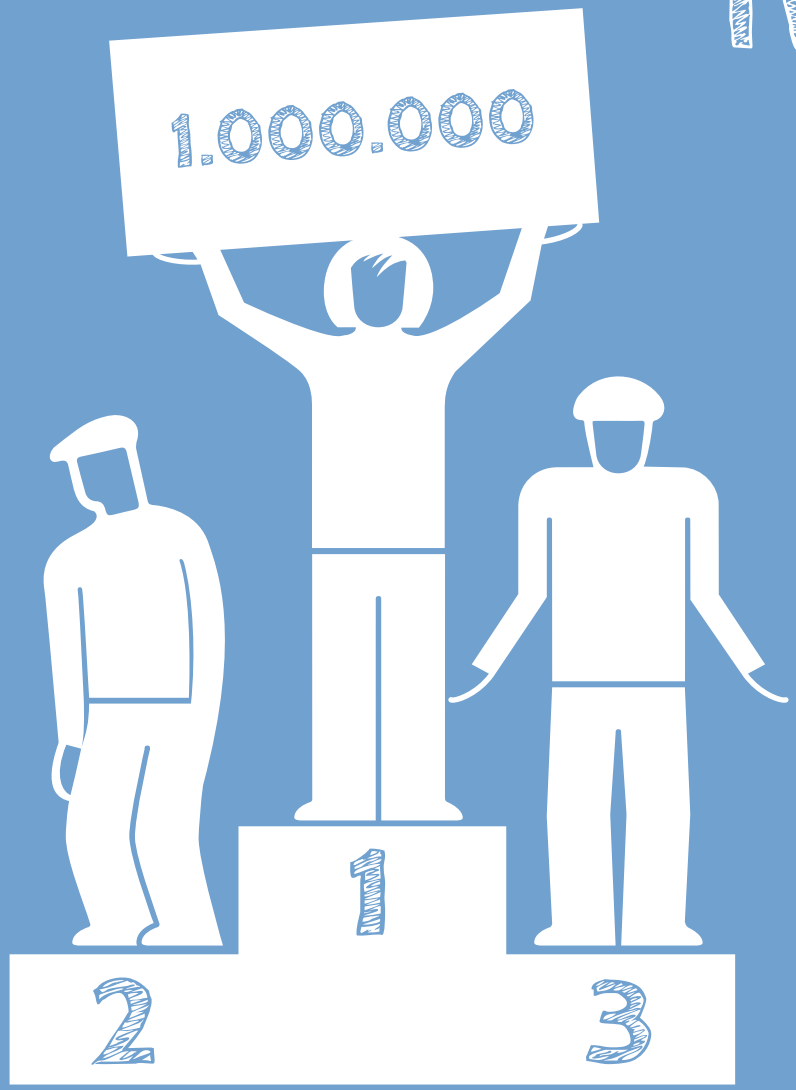
* The Lean Startup. Crown Publishing Group. Eric Ries

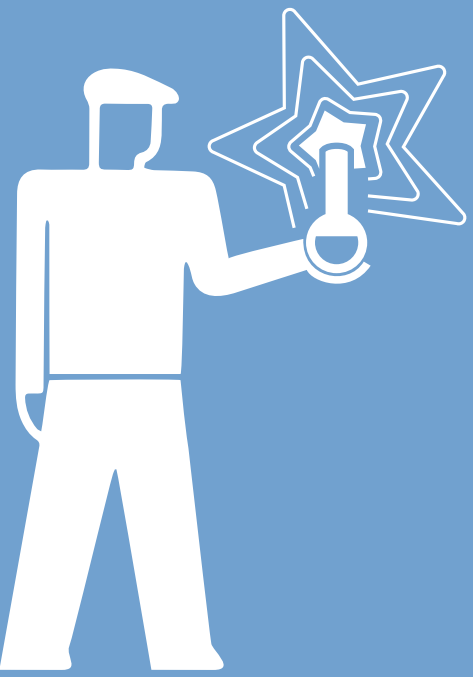
The co

Ways to find

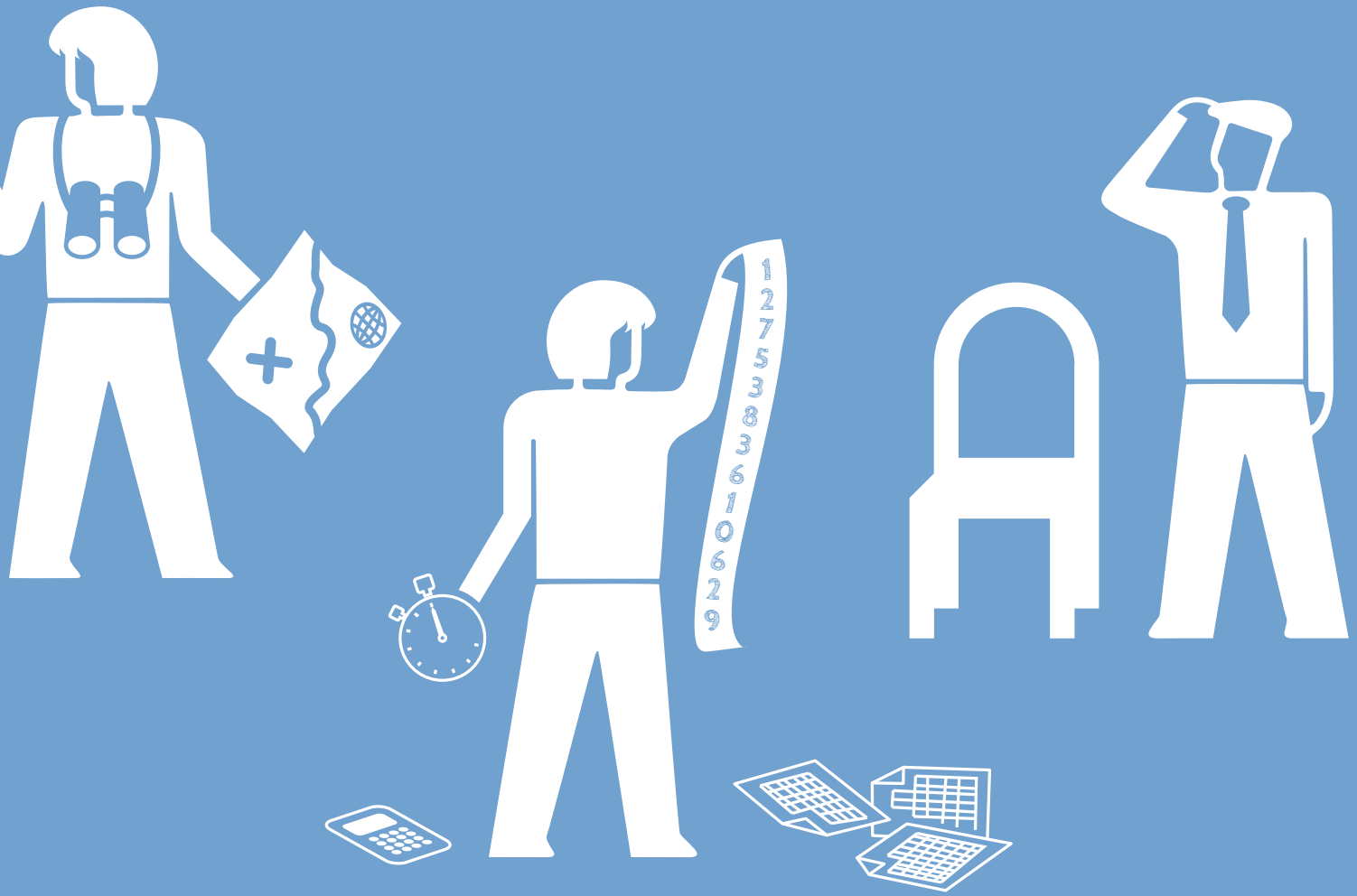

mpass
journeys

Two ways to





read this book



www.dbooks.org

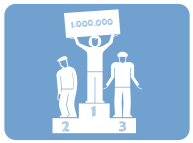
We didn't intend this book to be read from start to finish – but you are more than welcome to do so. Instead, we've written this book with busy managers in mind – people, like you, who don't have the time to read the whole book carefully but only the sections that are relevant to your particular business. We've therefore organized this book along a set of journeys.

The easiest way to read the book is to turn page and read through the brief scenario descriptions. So if you are interested in Open Source you can just read the two Open Source chapters.

A second way to read the book is to have a look at the five categories of drivers, presented to the right on this page spread. If you for instance want increased revenues, then there are two journeys that can help, Servitization and Open Source (business driven).



Drive revenue growth and outperform competitors with new business models



Journey 2
Journey 3

Develop innovative new products and services or improve current products and services



Journey 1
Journey 3
Journey 7

Expand into new markets and geographies



Journey 5

Increase quality, make OPEX savings and improve time to market



Journey 1
Journey 4
Journey 5
Journey 6
Journey 8

Deal with leadership challenges



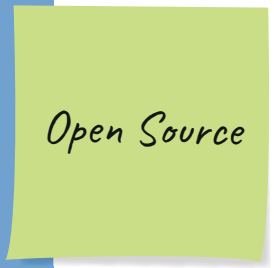
Journey 5
Journey 8

www.dbooks.org



Journey 1 - Co-operate in a community

It is admittedly an irresistible temptation to us, getting free software. The cost to maintain a particular part of our system is sometimes far too high. It is possible to develop common components in cooperation with others and gain from proven, de-facto standard software. Entering this path would encourage us to further advance how we do business.



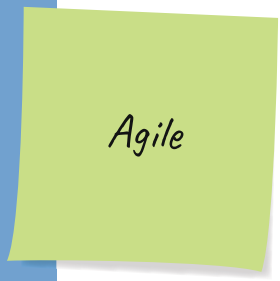
Journey 2 - Building ecosystems

By now we've been into Open Source development for quite a while. Even the management has acknowledged the contra productivity in just using free software without giving back, that sharing is caring. We imagine the ultimate Open Source strategy, to go beyond the communities and orchestrate our own ecosystem, to divide and conquer.



Journey 3 - Add supplementary services

Customers are getting more and more demanding: they want it all and they want it now, not to mention the fierce competition. Their offerings are basically the same as ours, equally or even better priced. Our product-oriented business slows us down. We need to move towards a service-driven business model.



Journey 4 - Deliver 24/7

Our customers are dissatisfied with our ability to deliver what they want and when they want it. The time we need to test and deliver makes it hard to meet deadlines and steals time from development. As if it weren't bad enough, serious bugs have started to pass the loupe. If we can deliver continuously, we can refine our services by running more experiments and do-learn-adapt cycles with our market.

Agile

Journey 5 - Pump up the volume

We have the greatest product; every batch we produce literally sells out as soon as it leaves the production line. We need to throw in more people, to build the products faster and increase the volumes. Innovation is not a matter at this point, neither is the cost. What matters is how to get around the many dependencies between products, services and departments.

Agile

Journey 6 - Agile and disciplined

As manufacturer in the automotive industry, everything we do need to adhere to industry standards. The Niagara-like waterfall processes makes even small things take ages. We would surely benefit from a more flexible workflow, but the OEMs kill every discussion by saying "Agile software development lack discipline."
We need to break this barrier.

Offshoring
Outsourcing

Journey 7 - Outside the box

How about moving parts of our software development to India? Learnings suggest that it might be difficult, that a cost-reduction isn't made that easy. Yet, highly skilled and talented engineers can still be recruited at a relatively low cost. Incorporated sensibly, we would gain back the flexibility we lack. We need to go offshore.

Basic
software
engineering

Journey 8 - First things first

It didn't happen overnight, but still, if we had staid calm when the business took off, we wouldn't have been in this situation. 15 additional programmers have joined the two of us, and our development process is getting a bit shaky. We need to adopt essential engineering principles and possibly re-factor the software architecture.

www.dbooks.org

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

The jo

Travel

© The Author(s) 2017
B. Fitzgerald et al., *Scaling a Software Business*,
DOI 10.1007/978-3-319-53116-8_2

urneys

brochures and stories

www.dbooks.org


And stories
60
86
100
120
142
160
182
212

Co-develop in a community

Scaling with Open Source

Building ecosystems

Scaling with Open Source

Add supplementary services

Scaling with Servitization

Deliver 24/7

Scaling with Agile

Pump up the volume

Scaling with Agile

Agile and disciplined

Scaling with Agile

Outside the box

Scaling with Outsourcing or Offshoring

First things first

Basic Software Engineering

Co-develop in a community





<Think big. How to transform a software business in a good way>

Copyright (C) <2016> <S.W. Scalare>

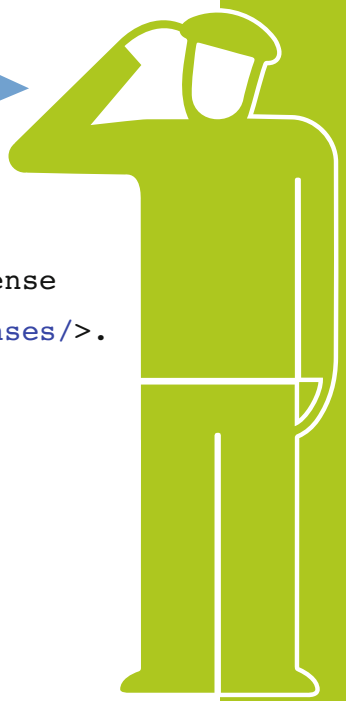
This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Open Source refers to software that has been made available to the public and is free to use in any application.

The source code is tied with a copyright license, giving any receiver of the source code the rights to use, modify and redistribute the code for free. Think of free as in free speech, not free beer*.



"Free software is a matter of liberty, not price. To understand the concept, you should think of free as in free speech, not as in free beer."

—Richard Stallman

The Open Source movement is several decades old, but it wasn't until the turn of the millennium that major companies entered the game. Traditional business wisdom had suggested that source code, which was seen as a "crown jewel" of a software company represented valuable intellectual property that should remain closed to maximize profit. With Open Source Software (OSS) development the effectiveness of this tactic is reduced since the source code is made publicly available.

There is always someone who knows the solution to a given software problem

The original intention with Open Source is that software is collectively developed, typically in an Open Source community characterized by collaboration, transparency and self-organization. This development model is an interesting value proposition to companies, as development is potentially done quickly, involving hundreds of developers, who work for free. Involving many people to fix defects gave rise to Linus's Law*: given enough eyeballs, all bugs are shallow. In other words, there is always someone who knows the solution to a given software problem. This may also lead to new and better ideas from someone who has stood on the sideline thus far—these "lurkers" may decide to contribute after using the software for a while.

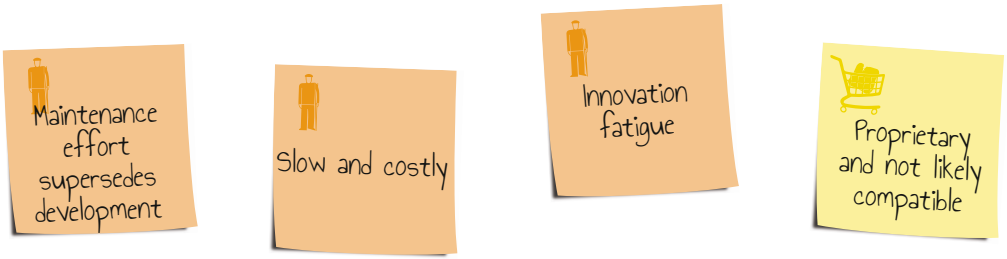


In recent years, this image of OSS developers has become outdated, and many companies are now actively participating in OSS communities for a variety of reasons.

However, no matter the business incentives, the initiative to start working with OSS communities is almost always taken within the development organization. As any development organization, they simply need to:

* A claim formulated by Eric S. Raymond, named in honor of Linus Torvalds

- Increase Innovation – the drive to include novel and innovative software in the solution, as well to participate in the communities were this innovation occurs.
- Reduce Time-To-Market – as a solution is available for free, and much of Open Source software has become de-facto standard, it can reduce the time for an offering to reach the market. More so, by active involvement in a community a company may influence its development to take a beneficial route for the company's offering.
- Reduce maintenance costs – sharing the development and maintenance as conducted in an Open Source community, substantially reduces the overall operating expenses.



Even if the work with Open Source preferably ought to be sanctioned by management it often starts as skunkwork*.

Those Open Source drivers are particularly appealing for the development organization whose software is characterized by a large heap of legacy and proprietary code that haven't been maintained and modernized for a long time.

When the cost to maintain the code supersedes what is invested in new features development, Open Source offers an irresistible temptation for engineering. Business as usual – to constantly postpone innovation and delay novelties in the offering to the future – simply isn't a viable option.

* A skunkwork project is a small and loosely structured group of people who research and develop a project primarily for the sake of radical innovation.

www.dbooks.org

So the engineers decides to "accidentally" use Open Source as a novel way to cut corners when solving development challenges they have at hand. This is very common, but of course not the preferred way forward. Quite likely, they don't have formal approval from the management.



The reason they don't include management and keep doing skunkwork may vary. The not-invented-here argument is often heard. Management may not trust third-party code. Until only a few years ago, Open Source was considered by most as a hacker's phenomenon and a headache for the Legal department. The threshold to explain what Open Source is really about and why software development would benefit from it might appear like an impossible mission. On the other hand most Open Source newbies are everything but knowledgeable about legal obligations that come with Open Source. Neither are they likely to comprehend how to truly leverage from Open Source as long as they only consider it as being "free as gratis" code. Both development and management need to learn about it.

Most optimally the use of Open Source ought to be introduced in a company as a coordinated and strategic activity, on which the following pages will elaborate in more detail.

One of the goals could be to get the entire organization seeing how Open Source saves money and improves innovation, to get it to use Open Source software more regularly. But foremost, control has to be established. The Open Source activities need to be agreed and standardized within the organization.

The company introduces policies, procedures, organization and hopefully some training as well to harness the Open Source practices, very much for securing the fulfillment of legal obligations that follows with Open Source.



However, introducing Open Source in an organization raises some concerns too. Training all staff on the fundamentals of copyright law and Open Source licenses is costly and might offer a challenge for engineers to grasp. It could be hard to implement and defend the added cost for the necessity of conducting the Compliance work that follows with Open Source.

Awareness on costs related to Open Source arises, both as a threat (potential litigations for license breaches), but foremost as an opportunity (faster development and reduced maintenance). Gaining the benefits of the latter will open for allowing developers to engage in Open Source communities, including source code contributions, as that will soon will be discovered that is the only feasible way to influence Open Source communities.



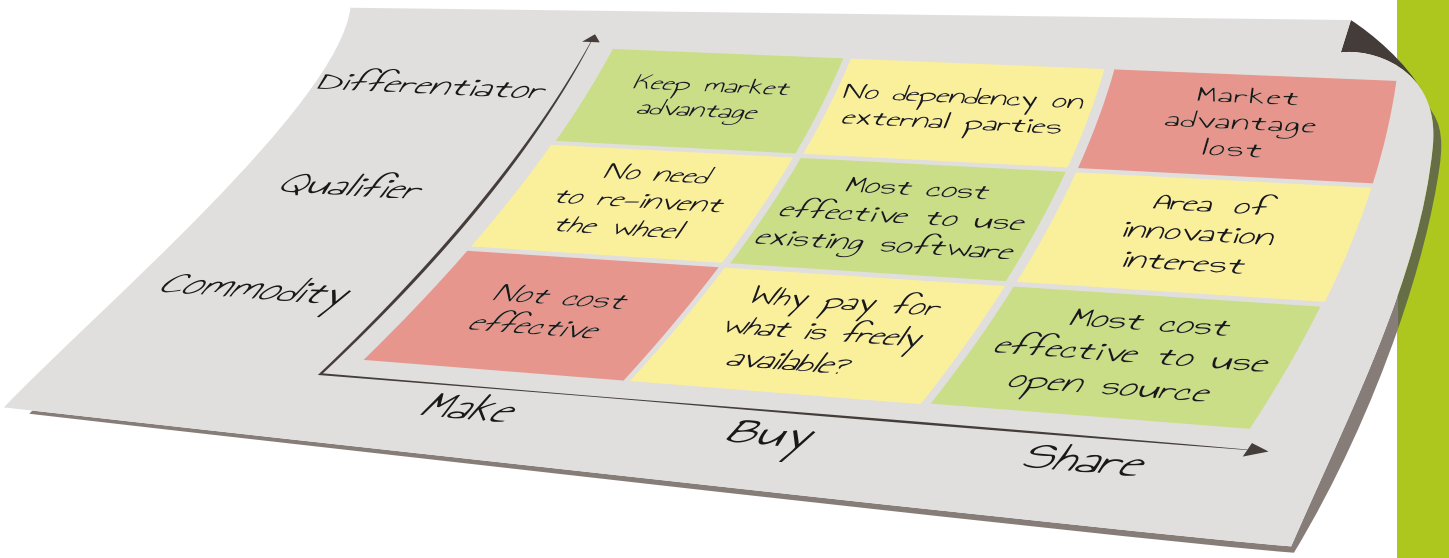
Initially, the likely main challenge for management is that Open Source is perceived to raise an unknown legal risk. Management may also perceive that with Open Source unknown technology, with unknown effects on the company's offering, would flow in uncontrolled, and worse, that valuable corporate Intellectual Property may flow out uncontrolled. Typical legal risks include copyright and patent lawsuits and to lose control of software and other intellectual property rights. In fact, to handle these matters comes at a bargain in relation to what we gain, compared to continuing the business as usual. We just have to deal with it properly.



Open Source governance policies, processes, tools and organizational structures will be required. Three fundamental processes are essential; an Intake process for legally vetting code that development intends to use, a Compliance process for ensuring that code follows the terms and conditions of Open Source licenses, and a Contribution process for the approval of code to be released to an Open Source community.

> We will need organizational roles such as the Open Source Officer and specific roles for compliance and contribution management.

These processes will call for creating new organizational roles such as the Open Source Officer and specific roles for management of compliance and contribution. The new roles will have to



possess mandates that management might be unwilling to share. An implication that is seldom fully understood is that the current product management has to let go parts of the requirements and the product definition. By its very nature, the power over the product definition will drift toward the engineers and the Open Source communities.

Getting control of the legal matters requires legal and patent counselors to be involved in the software development process. The counseling will be around copyright, patent, IP, and trademark laws. As contributions to Open Source communities become frequent, a larger organization could benefit from establishing a governance body, a so-called Open Source Board. Typically such an Open Source Board vets and approves contributions, while considering both business needs and IP protection needs. Such a body would also naturally be mandated to issue corporate policies on Open Source.

www.dbooks.org



To most developers this goes without saying: Integrating Open Source software is no different from integrating any 3rd party software. The software architecture has likely to change to host the Open Source software artifacts. If it's already modular, just with a few cuts in the wrong places, the adaptation will be a smooth job. If it's monolith, a major refactoring of the software is likely required. The Open Source software's APIs must in turn never be changed without approval, to facilitate contributions back to the Open Source community.

To some companies, the Open Source software becomes key elements in both the company's offering as well as its innovation. The companies' engagement in Open Source has to get directed and has to evolve to an Open Source strategy that encompasses development goals as well as business benefits. They need at least to influence the community to gain some control of the development in the community. The incentives to do this ranges from simply sharing development costs with partners and use common technology, to give the company a competitive edge on its own offering while disrupting the competition. In most Open Source communities the control is i gained by becoming a champion contributor.

Introducing Open Source development sensibly and cross the organization as a strategic and coordinated activity, will likely pay off in a more competitive product offering. Getting state-of-the art Open Source technology to a reduced cost and with a shorter lead-time isn't that bad, after all.



One more thing

In many large companies, software is considered the property of the team that developed it and is not shared freely within the company. **Inner Sourcing** is the practice of developing and sharing software openly within the company but not beyond (in contrast to Open Source Software).

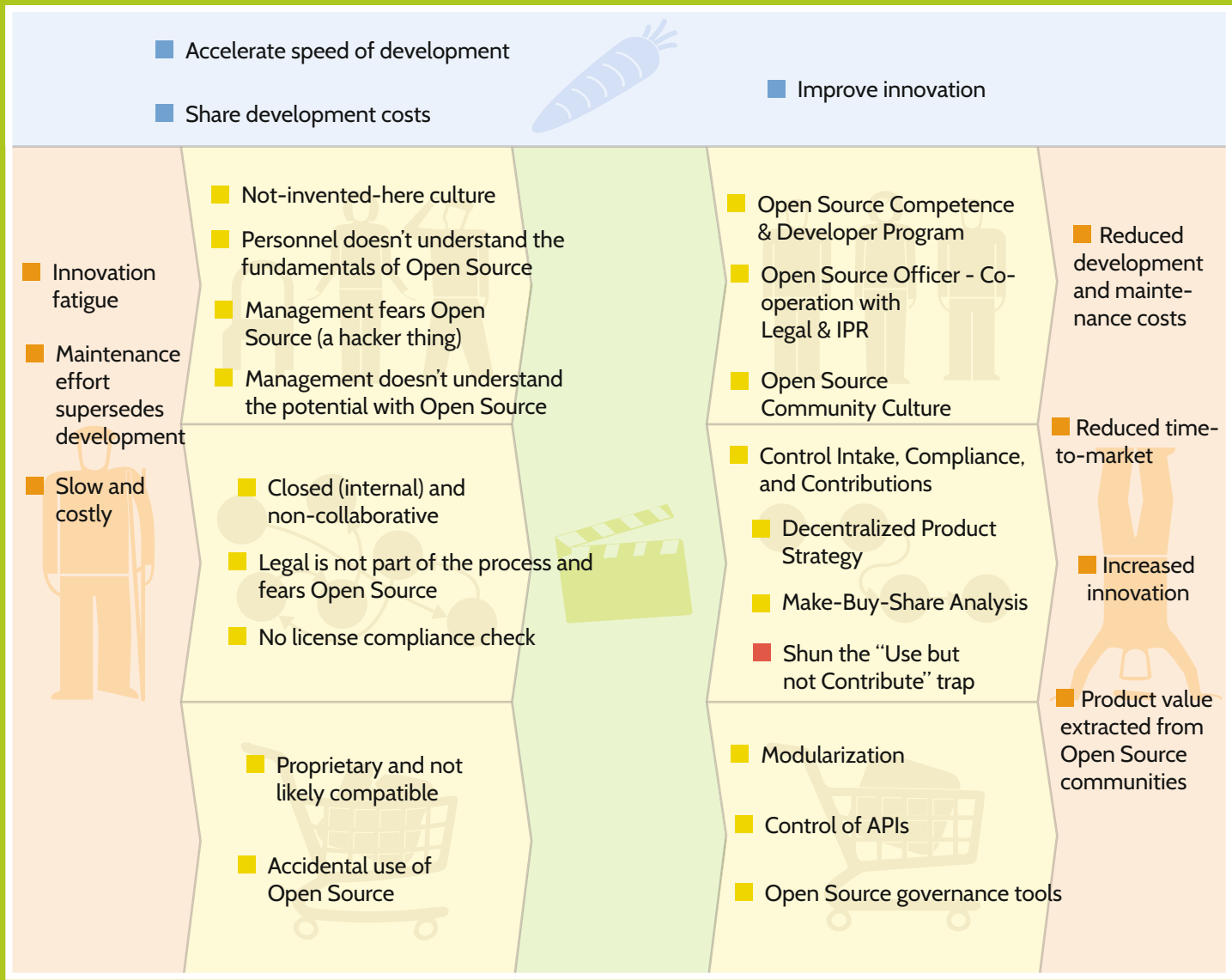


The companies get many advantages of Open Source software and avoid at the same time IP and license complications associated with open source software contributions. The primary benefit is however the collaborative spirit and organizational flexibility that arises from teams who are able to contribute improvements to each other's code. In addition, there is a direct benefit in reusing code and competence, something that will not only reduce development cost but also increase speed and overall quality.

Volunteer contributors will spring up freely to contribute to interesting projects

There are many other reasons to consider Inner Sourcing. An open environment facilitates increased awareness of the software and helps to break down barriers between teams through the use of common code, tools and methods. Having less duplication of development will cut costs. Volunteer contributors will spring up freely to contribute to interesting projects, which may lead to shorter time-to-market. Since contributions are under large-scale scrutiny, developers get aware of their reputation and motivated to write "good" code.

Moreover, since they are familiar with a standard set of common tools and infrastructure, developers can be more easily transferred to other projects or products. This in turn will reduce time-to-market, as project start-up time can be reduced.


Accelerate speed of development
Share development costs
Improve innovation
Innovation fatigue
Maintenance effort supersedes development
Slow and costly
Not-invented-here culture
Personnel doesn't understand the fundamentals of Open Source
Management fears Open Source (a hacker thing)
Management doesn't understand the potential with Open Source
Closed (internal) and non-collaborative
Legal is not part of the process and fears Open Source
No license compliance check
Proprietary and not likely compatible
Accidental use of Open Source
Open Source Competence & Developer Program
Open Source Officer - Co-operation with Legal & IPR
Open Source Community Culture
Control Intake, Compliance, and Contributions
Decentralized Product Strategy
Make-Buy-Share Analysis
Shun the "Use but not Contribute" trap
Modularization
Control of APIs
Open Source governance tools
Reduced development and maintenance costs
Reduced time-to-market
Increased innovation
Product value extracted from Open Source communities

Get inspired

This scenario has been based on case studies of different companies that have made this journey, to scale with Open Source. Learn from their experiences, what they gained and what they had to overcome.



Sharing is caring

A quiet revolution was changing the world for the software engineers at the mobile phone manufacturer Sony Ericsson. An Open Source force of epic dimensions had just been released. This case study is about the manufacturer who embraced a powerful engineering movement and created an Open Source strategy that, although there were initial business benefits, primarily encompassed goals set by the developers.



A thriving Open Source culture behind the wall

Open Source projects really can't be beaten in terms of development power. This case study gives insights into a multi-national mobile network equipment supplier, which experienced a similarly quiet but powerful Open Source revolution, but behind the company firewalls. Read about how the development organization found ways to utilize resources and created an Internal Source culture.



Keeping the doors open

Traditionally, door-opening solutions were about locks and keys. Nowadays, they are still about mechanics but use a lot of electronics and software. It is not just a piece of hardware anymore. The solutions even tap into the industry of services. And there is of course Open Source software usable also in this particular business. Read about ASSA ABLOY, who went from using bits of Open Source software they found, to get really professional about it and implement an Open Source strategy cross their organization.

www.dbooks.org

CASE STUDY / Co-develop in a community

Sharing is caring



Sony Mobile's Open Source journey began with general curiosity among developers at Sony Ericsson's development department. This led eventually to an investigation whose results were so convincing that the management took the bold decision to shut down the work on proprietary operating systems and invest wholeheartedly in the Open Source based Android operating system. The new era began in 2010 with the Android telephone Xperia X10, and today all its smartphones are based on the common Android platform.

The company sees a whole host of advantages in belonging to the Open Source movement. First of all, it increases the pace of innovation and development, simply by having a larger number of developers focusing their creativity on a single task. And when someone comes up with a clever solution, the entire network has a share in it and benefits from it. As a result, everyone is constantly creating new opportunities for everyone else. Without Android, Sony Mobile don't believe it would have existed. More than 85 % of its software is based on Open Source.

It realized early that an Open Source project must be transparent and encourage participation and collaboration. Management had to make Open Source a part of their corporate culture.

R & D and Legal, hand in hand

Close collaboration with the Legal department has played a central role. A success factor was the early establishment of a "double-command", consisting of its chief strategist for Open Source and a lawyer at its Legal department. The duo has been instrumental in changing the business and mindset throughout the development organization.

The Legal department recognized in the initial stage that Open Source was perfectly solid and valid from a legal standpoint – acquiring Open Source was essentially the same as any other kind of third party software. They also noted that Open Source would become utterly essential for the company's survival from a business perspective. In this way, the Legal department became a key player in persuading executive management that the necessary culture shift not only was possible, but also would be warranted by governance under Legal's supervision.

www.dbooks.org

A major challenge was to fight the notions that Open Source was mainly a software development concern and something of slight headache to the Legal department. These earlier viewpoints dominated the thinking in Sony Mobile's early days of Open Source. It was seen as OK to use because it was "free of charge". In the last couple of years, as the success of Android unfolded, the mindset changed completely.

Eventually, software developers and lawyers developed mutual trust. The duo translated legal concerns to developers, as well as the other way around, educated Legal on engineering concerns. The resulting trust has also led to executive management being much more daring in taking business initiatives involving openness and collaboration – even when it involves the competition.

Sony Mobile Open Source Maturity and Strategy model

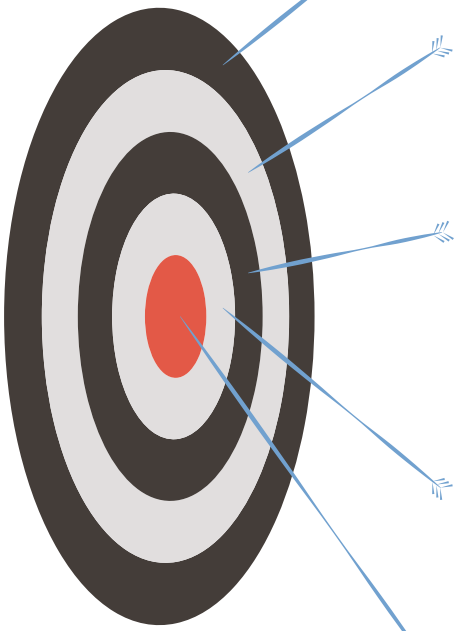
Today, not only does most of Sony Mobile use Open Source for everyday development, but the company has also established itself as the largest external contributor to Android development.

Recently, it has also taken several initiatives in the marketplace in leveraging Open Source in tilting business to its advantage. It is making progress in achieving position on the higher levels of the Sony Mobile Open Source Maturity and Strategy model, the levels when business concerns come into play.

Its Maturity and Strategy model has five levels. The first three represent stages that are mostly driven by engineering and development concerns, whereas the last three levels are more business driven.

Even if the model isn't a tool for scaling into Open Source development, the model has proven to be very effective to communicate where it is and where it aims. It's a model to measure maturity and to set the strategies that works with both developers and management.

Engineering Driven



Level 1, "Accidental"

A stage of discovery and early awareness where developers explore and "play around" with Open Source software.

Level 2, "Repetitive"

A company begins to make use of Open Source software in a governed way, seeing that it can reduce cost and improve interoperability.

Level 3, "Directed"

Open Source has support from executive management, is incorporated in the product strategy, development aims for champion communities and collaborations widens.

Business Driven

Level 4, "Collaborate"

Open Source projects and collaborations are run to achieve both technical and more so business goals which are able to change the market logic.

Level 5, "Prevail"

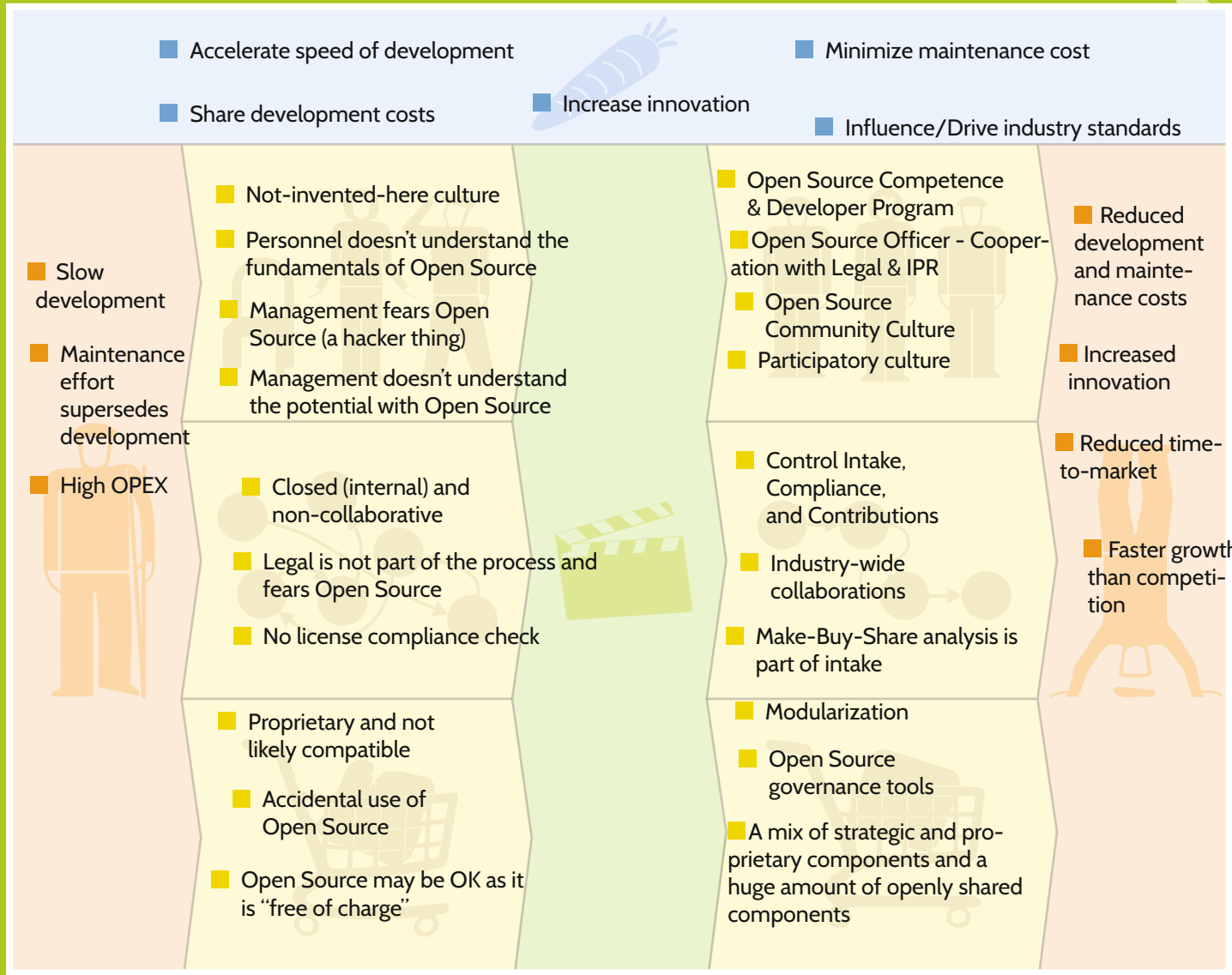
The company has developed a fully-fledged Open Source company culture, with full strategic support from the top management, to the extent the company is able to disrupt entire markets.

www.dbooks.org

Advices & take aways

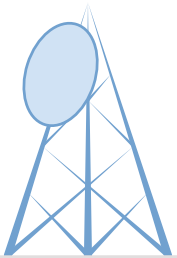
- **Use an Open Source Maturity model**
  Understand the level of maturity to drive change, provide a vision and outline a strategy to drive change. Use this extensively as a communication and education tool. Rules and roles regarding project contribution are important – Inner Sourcing projects should start with defining contribution rules and responsibilities.
- **Describe the processes for governing Open Source**
  This should include how we work, our roles and our responsibilities. The most important processes are intake, compliance and contribution.
- **Take benefit from tools**
  Take benefit from an Open Source audit tool to ensure compliance and to extract copyleft* code. Though, it's important to recognize that the main benefit of a tool is to reduce engineers' workload – a tool can't itself replace a lack of policies and processes.
- **Educate, educate, educate**
  In addition to courses, an everyday present spirit of education should be a part of all interaction. Lead by example, following the three key concepts of transparent, participative and collaborative.

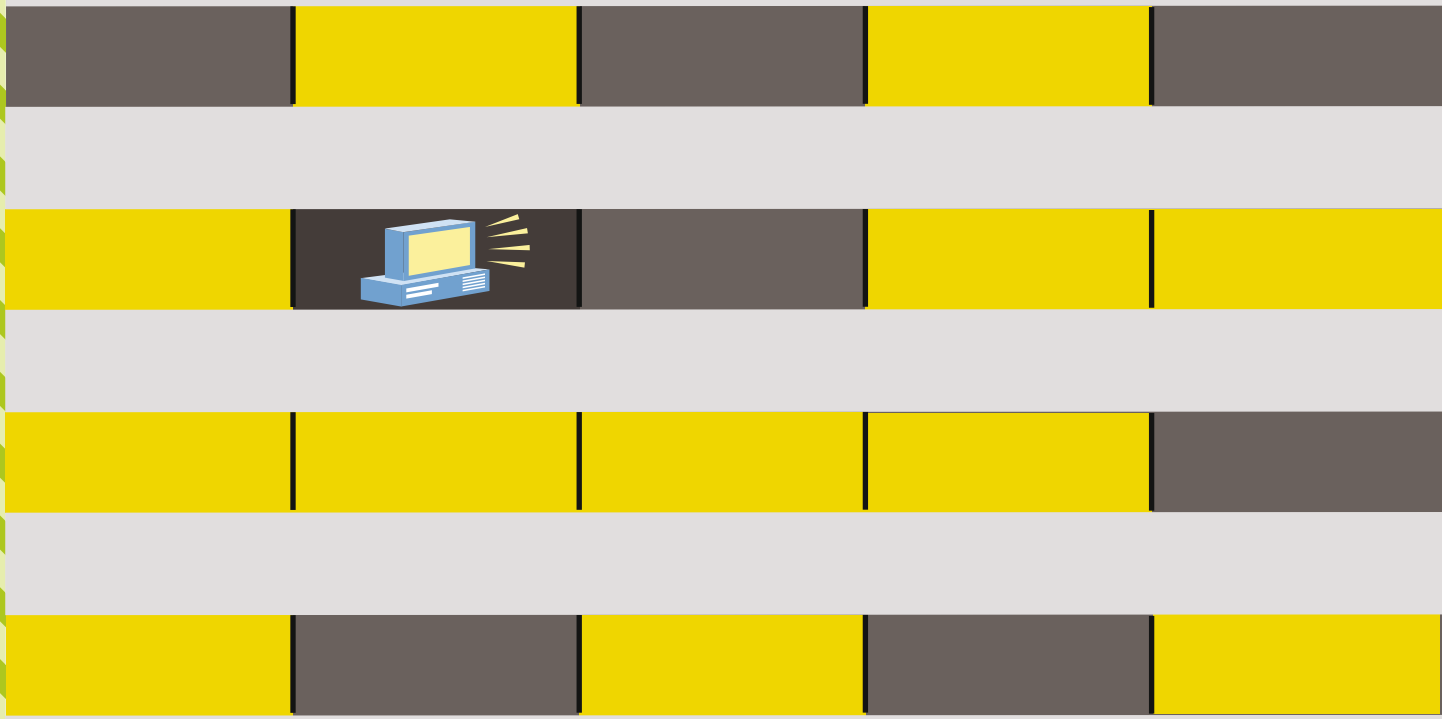
* Copyleft (a play on the word copyright) refers to the copyright licensing scheme used when making Open Source contributions.



www.dbooks.org

A thriving Open Source culture behind the wall





Adam is a senior software developer at a big Swedish company in the telecom business. Some years ago, Adam realized that many projects and developers more or less developed the same code. There had to be a way to reuse code and competence across the company. Since he was already familiar with Open Source software he thought a similar approach would also work internally. So he made his own software repository solution available to his colleagues so that they could start sharing code and projects.

The rumor spread and more and more colleagues started to use Adam's tools for sharing code. Later, several teams decided to manage all their internally developed tools by using this repository installation, which still was running on Adam's PC. This was just the start. By the word of mouth and some internal blogging, the initiative spread within the company and grew to such proportions that also other managers realized their organizations depended on Adam's code sharing initiative.

The IT department had at the time licensed a commercial system for code life cycle management but it turned out that the developers rather desired to use Adam's tools. A few attempts were made to get developers to use the commercial solution, but Adam's solution was already too well established within the organization. 5 years later, the Inner Source initiative had grown to a staggering 5.000 projects and 16.000 people. And the only support Adam provided was some short documentation on how to use the tools and some thoughts on Inner Sourcing through his blog.

The company has since long realized that this was a too important and powerful environment to be managed and driven by one person on a small PC. Eventually, they also switched to use a standard commercial system for code sharing and collaboration.

The main reasons the Internal Source initiative grew so strong in the company are the low entry barrier of the tools and the extremely short lead-time for starting up a new project. Developers like it because it is so easy to use and that they instantly get up and running.

Internal tools are very often first out to be developed in an Inner Sourcing environment. Adam estimates that their internal tools development became 10 times more efficient in terms of resources, by this initiative. A side effect, but a very powerful one, was that the company also

www.dbooks.org

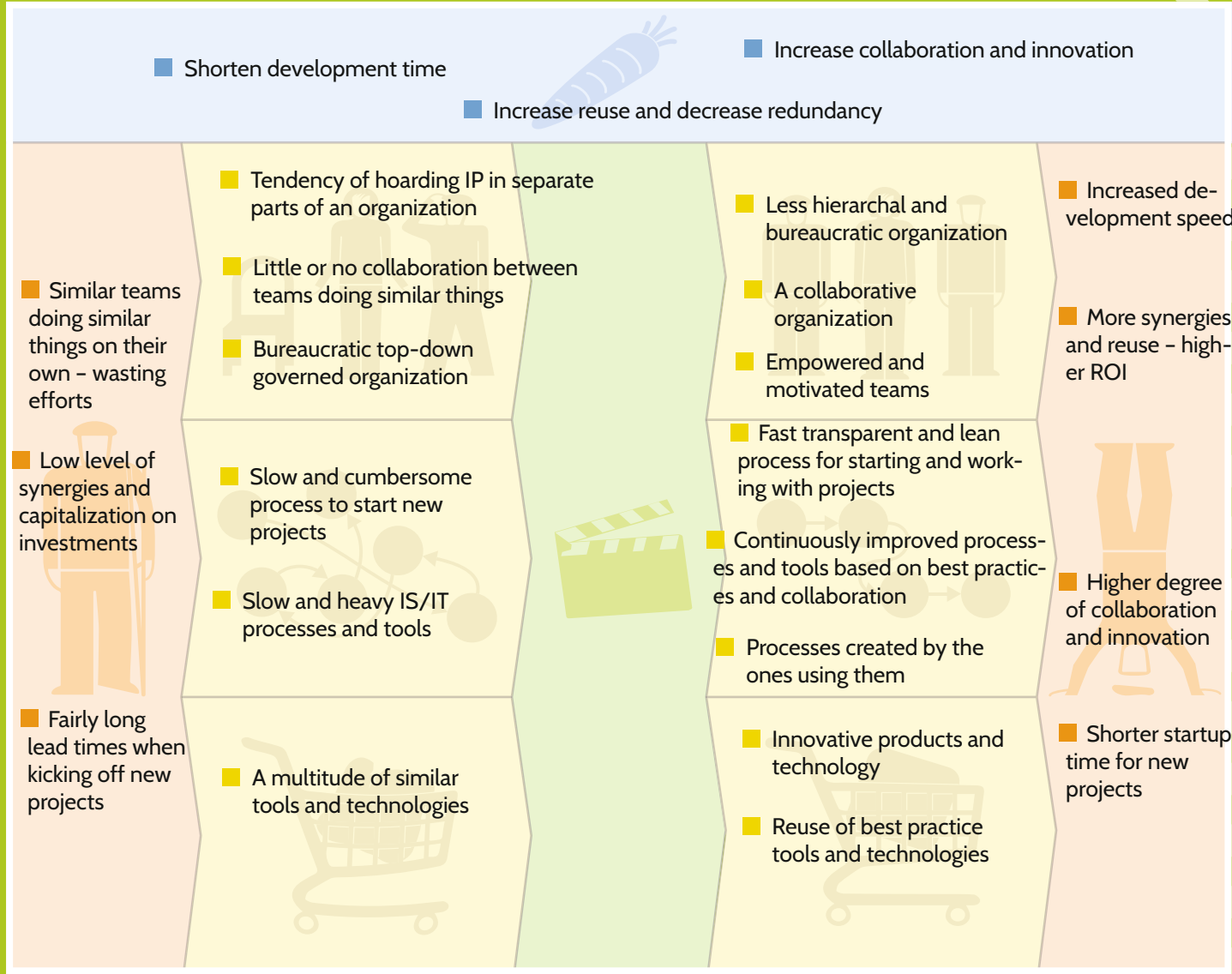
gained full transparency in development made by third parties, since they as well started to use the Inner Sourcing environment.

A key success factor behind introducing Inner Sourcing was the corporate culture. The teams are fairly autonomous and well empowered to define and use their own methods and ways of working. Adam states that to succeed with Inner Sourcing, the company culture has to be built on trust and empowerment of teams and individuals.

Even though tools still are the most common Inner Source projects within the company, also a few commercial products rely on Inner Sourcing. Adam is convinced that Inner Sourcing will be instrumental to tackle future capacity and time-to-market challenges.

Advices & take aways

- Culture is extremely important – trust, openness, collaborative and empowerment are key ingredients.
- Rules and roles regarding project contribution are important – Inner Sourcing projects should start with defining contribution rules and responsibilities.
- Financing of Inner Source projects is a common challenge and not always easy to solve. It's best to settle this before starting a project.
- Review and evaluate different tool suppliers carefully before running out and purchasing tools and systems


Shorten development time
Increase collaboration and innovation
Increase reuse and decrease redundancy
Similar teams doing similar things on their own – wasting efforts
Low level of synergies and capitalization on investments
Fairly long lead times when kicking off new projects
Tendency of hoarding IP in separate parts of an organization
Little or no collaboration between teams doing similar things
Bureaucratic top-down governed organization
Slow and cumbersome process to start new projects
Slow and heavy IS/IT processes and tools
A multitude of similar tools and technologies
Less hierarchal and bureaucratic organization
A collaborative organization
Empowered and motivated teams
Fast transparent and lean process for starting and working with projects
Continuously improved processes and tools based on best practices and collaboration
Processes created by the ones using them
Innovative products and technology
Reuse of best practice tools and technologies
Increased development speed
More synergies and reuse – higher ROI
Higher degree of collaboration and innovation
Shorter startup time for new projects

Keeping the doors open



One very clear example of a company that has moved from hardware-based products to software-intensive solutions is ASSA ABLOY. This company is a leading manufacturer in door-opening solutions and a market leader in Europe, North America, China and Oceania. The company was formed in 1994 through a merger of ASSA in Sweden and Abloy in Finland. Since then, it has grown from a regional company to an international group employing a workforce of over 46.000.

Traditionally, door-opening solutions were about designing and manufacturing locks and keys, and as such, this business is very much dependent on the price and availability of steel as the raw material. While steel and mechanical solutions are still important, most of the costs of the company's solutions are spent on software development. Modern door opening solutions involve a considerable amount of electronic circuits and software to control them. Furthermore, door-opening solutions now also start tapping into the industry of services. The days where a lock was just a piece of hardware are over.

ASSA ABLOY has developed software for decades for its back-end door-lock systems and Open Source isn't a new thing for them. Indeed, ASSA ABLOY were well aware of the benefits that Open Source Software can offer, such as reducing development time, increased security (as some of the relevant OSS components are thoroughly tested), and high quality products. However, there was never a company-wide strategy or policy around Open Source. As is common in many companies, different engineering teams first brought in Open Source in an uncontrolled way. As the company realized the increasing strategic importance of Open Source, the company started investigating tools and policies for using Open Source more consistently. This new task was assigned to ASSA ABLOY's Shared Technologies division, which is responsible for developing common software assets and scouting new technology.



ASSA ABLOY Shared Technologies understood that becoming familiar with the various Open Source licenses was key in order to become compliant. To that end, a tool was used that automatically searches and identifies Open Source software in their code base. However, soon the company realized that merely using a tool wasn't sufficient to engage with Open Source products. Engaging in Open Source also requires developing an understanding of the Open Source software lifecycle management, and how to align this with the company's internally

www.dbooks.org

developed software. A new policy were needed before making significant investments in tools.

ASSA ABLOY took a number of steps to establish an Open Source engagement policy. First, the company sought advice from Open Source consultants to be better informed about the consequences of using Open Source software. Second, prior to rolling out the newly defined policy and process, all engineers, project managers and line managers in the Shared Technologies division were enrolled in a training program on these issues.

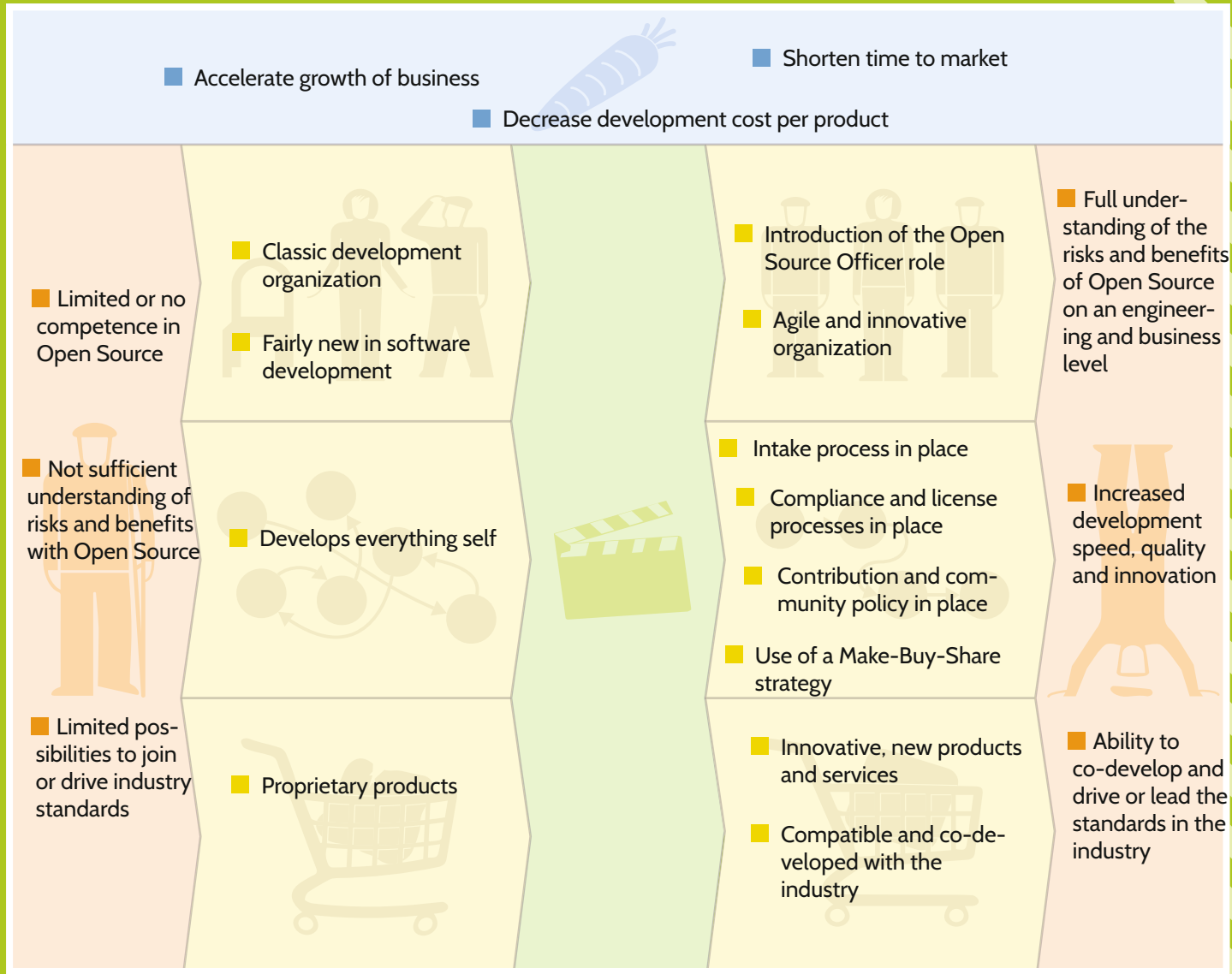
The company's process and policy defines a structured way to manage different Open Source software, which includes the following key processes:

- Acquisition. This process is concerned with avoiding risks related to IPRs, patents and security threats. Furthermore, it considers understanding the availability of Open Source software and developing a make-buy-share strategy for adding software components in their products.
- Compliance. The Compliance process deals with license matters and how their proprietary software and new Open Source software can coexist.
- Contribution. The contribution process defines how the company should interact with Open Source software communities.

A key challenge was really the absence of a company-wide Open Source policy. It was very difficult to leverage and get control of Open Source Software without it. Another challenge, one shared with many other companies, is how to more actively engage with Open Source communities. Even though the company management encourages participation in communities, the development teams are still mainly users, not contributors, of Open Source software. Achieving this is truly a challenge.

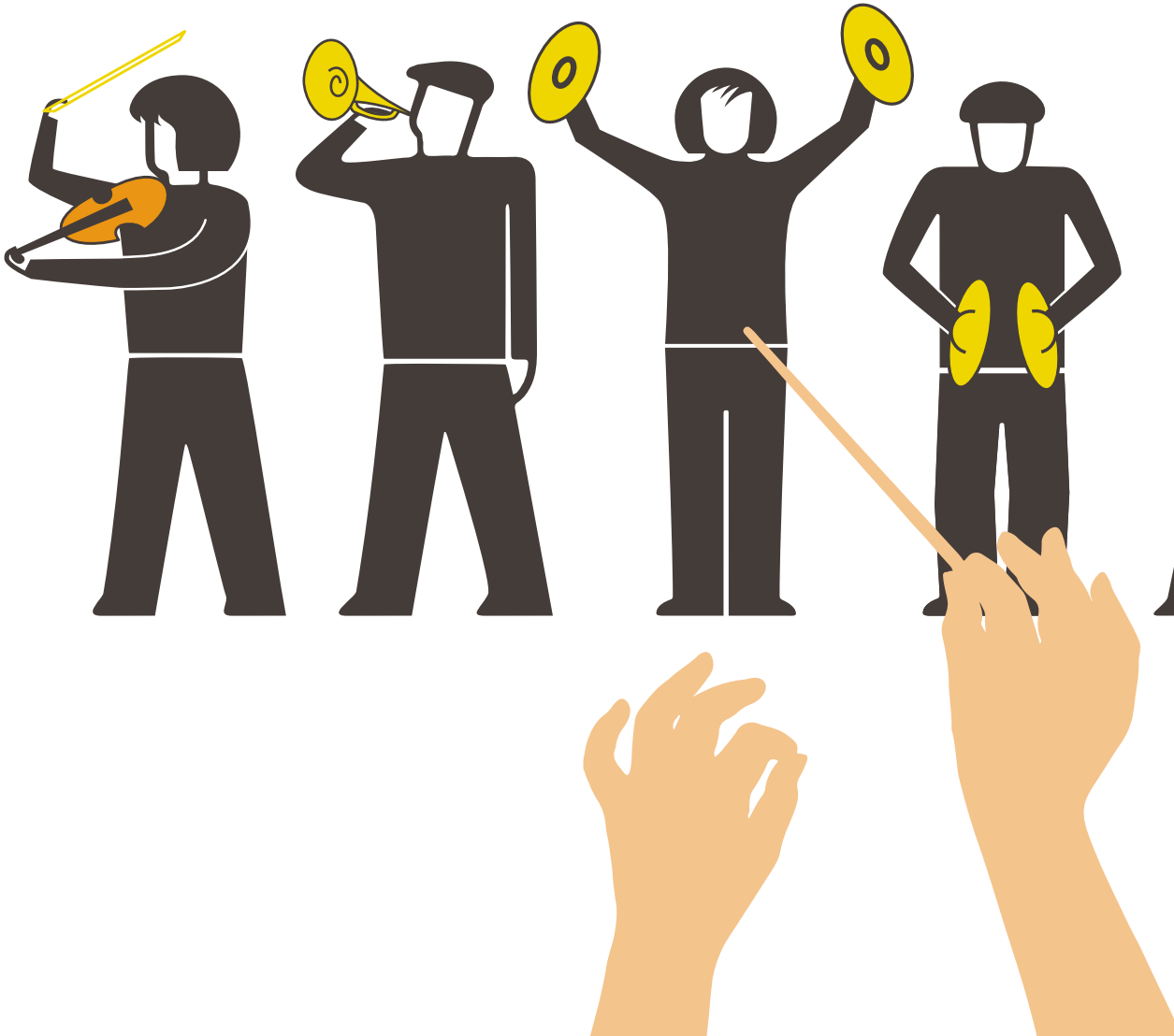
Getting the teams more involved in communities is truly a challenge

ASSA ABLOY Shared Technologies is clearly on a journey where it not only sees the benefits from using Open Source in their products but it also sees the value in actively engaging in communities, in order to actively participate in the evolution of Open Source products that it has a stake in.



www.dbooks.org

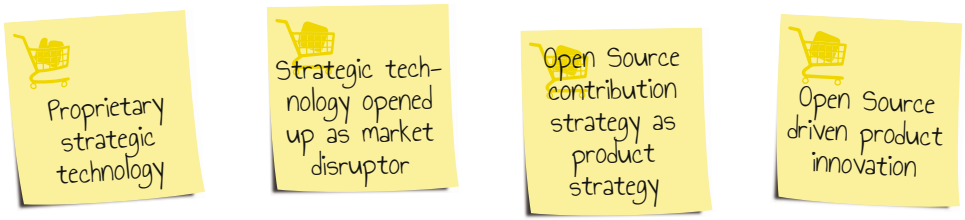
Building ecosystems





This scenario is about how to go from "only" creating product value from an Open Source community to drawing comprehensive gains from orchestrating an own ecosystem.





The ultimate Open Source strategy is to go beyond the communities and create an ecosystem that becomes the business and not just supports the business. At its furthest implementation, the ecosystem has disrupted the entire market logic and became the helm of the market. Google and Apple are great examples of companies that made exactly this. Google's Android is basically nothing but a collection of 60 to 70 Open Source software packages. Similarly, Apple's OS X and iOS are highly dependent on Open Source (BSD Unix and Web Kit among others). The key aspect of their success has been their ability to join together Open Source communities and blend differentiating (often proprietary) parts of their products with commodities offered as Open Source.



Open Source is a game changer to the business, since parts that are contributed as Open Source also gets commodities and brings down the value of the offering. The business drivers are mostly a product of the change in the business model. Typical goals that companies aim for are basically to:

- Accelerate growth of business – to expand the market both in terms of a broadened offering and to grow higher in the value chain
- Disrupt market entry barriers – to gain access to a market with an open offering, while raising the bar for proprietary and non-collaborative businesses
- Open up for alternative business opportunities – to benefit from alternative revenue streams since the revenues for software alone disappears

To achieve this requires great flexibility in creating new revenue streams. They can be based on an extended business model (when alternative revenue is collected from something related to the core offering, e.g. a service fee), an indirect business model (when revenue is mainly collected through a device or a hardware offering) or an asymmetric business model (when revenue is collected from a source unrelated from the core offering). An example of the latter is Google's Ad-Search business that benefits from providing Android for free.

Open Innovation (OI) is the dominant model of gaining external benefits and generating spin-offs from openness.

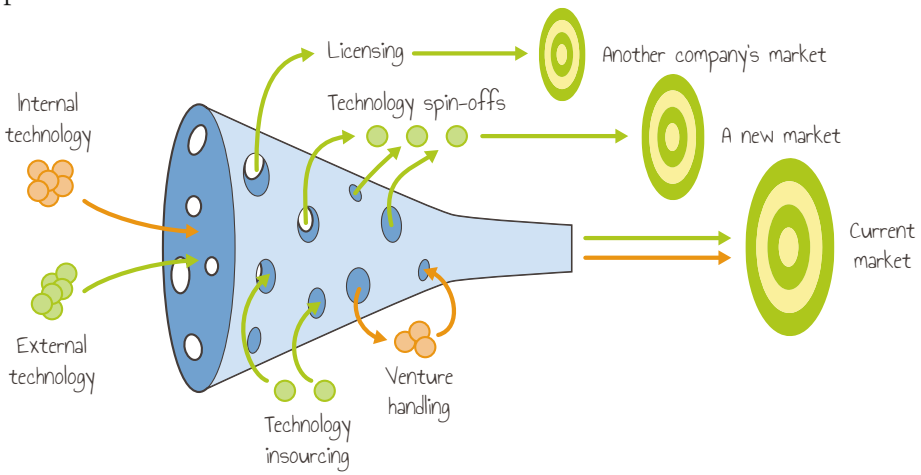


Illustration based on picture from presentation of Open Business Models by Henry Chesbrough

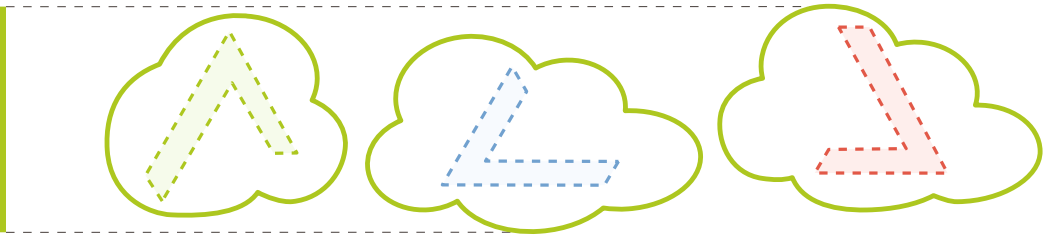
www.dbooks.org

The key desired abilities we strive for – to be able to reach, create and orchestrate an Open Source based ecosystem, an ecosystem that also could be described as a community of communities – require a collaborative business model.

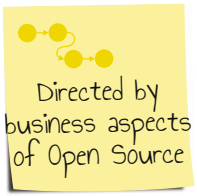


However, a journey like this can only start once a company's use of Open Source can be said is being directed, when the company has gained such support from the executive management so the use of Open Source software is paramount of the product management. (See the chapter **Co-operate in a community**.) At this stage the company is well versed on the engineering and legal aspects of Open Source. The company's knowledge level on Open Source is satisfying to the extent that directives and policies are relaxed and some automation of the governance processes has been introduced. Moreover, Open Source technologies have become such a valuable element of the offering it is necessary to influence the control of the Open Source development. The product offering itself has also gone through a transformation. Fundamental is that it has a modular architecture, but more, it's likely that the product is connected to the Internet, preparing the offering to be extended by cloud-based services.





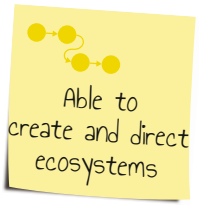
Although the entire idea with building an ecosystem is to scale the business, it can still be a challenge for the management to recognize how Open Source as a phenomenon can be used as a business tool. The management will have to explore radically different market logic, and in this, they have to fundamentally question what really is the company's core offering and how alternative revenue streams then can be created. The cemented truth that "proprietary Intellectual Property is paramount for a company's success" will often be proven wrong.



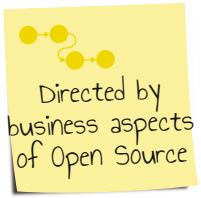
Most essentially comes with Open Source a participatory culture. Customers are moving from being passive receivers of purchased solutions to actively involve themselves in the development of the product offering itself. Many companies have created Developer Programs to catch and nurture this kind of culture, opening up for customers to influence the development of the product offering.

A Developer Program is very often seen as the seed of an ecosystem.

To better reflect and support the transformation to a highly collaborative business model, a more supporting and coaching leadership style is highly promoted. The business lends itself to be run by a flat, network-oriented and self-managed organization, which better reflects an ecosystem structure.



www.dbooks.org



The current product Management will obviously be under fire since much of the product definition will reside outside the organization. It's important though, that the role for the product Management is adapted to envision future offerings rather than define the product in detail, as the latter will drift towards crowd-based requirement engineering set-up in the ecosystem.

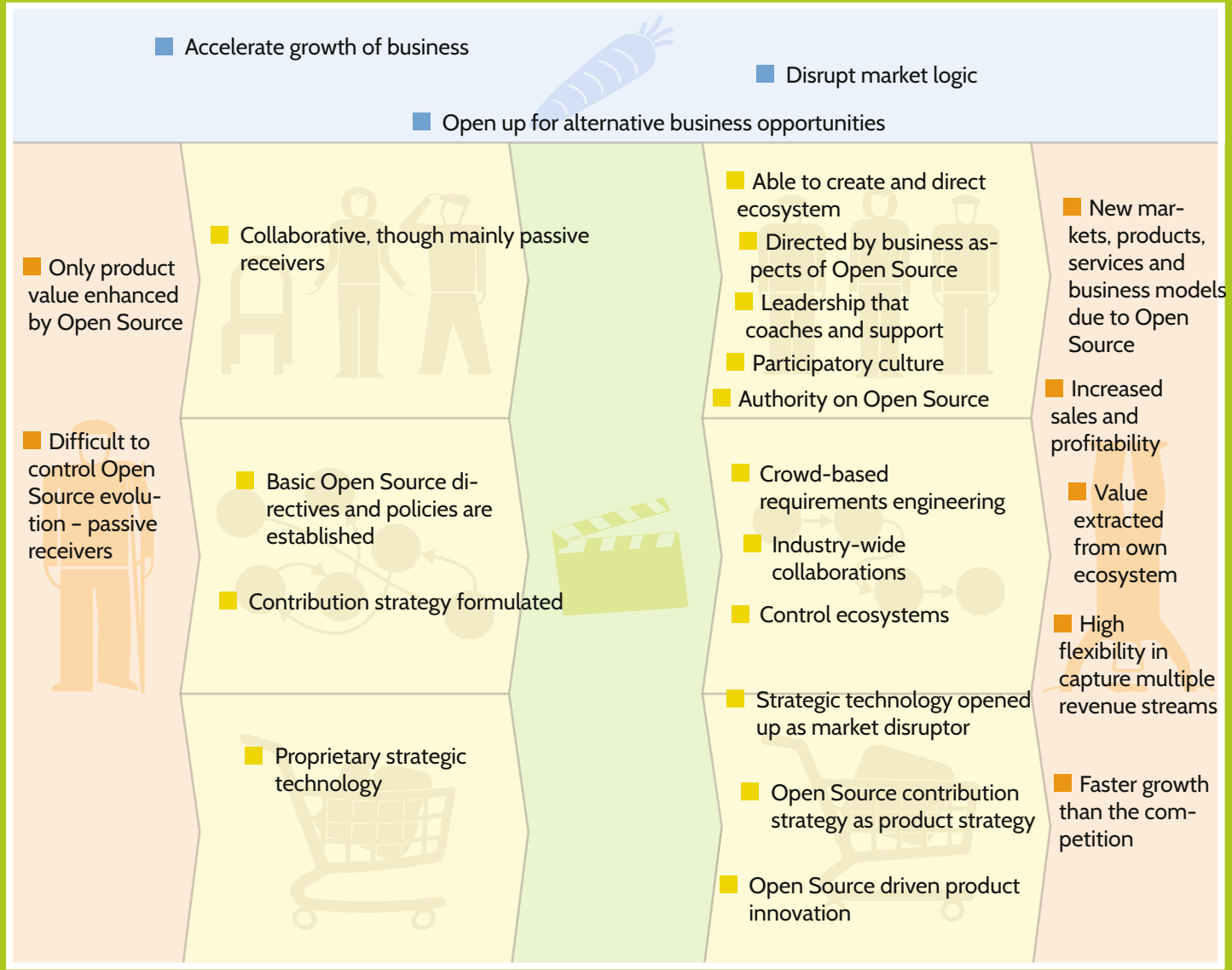
The Legal department will deepen its collaboration and start building legal frameworks for novel offerings, as part of the company business development.

However difficult the transformation of management might appear, a major growing pain will be to recruit the necessary talents in order to build the ecosystem. This accounts in particular for cloud-based services and the necessary support systems for the new business it will offer. Those people are currently amongst the most sought after in the industry.





To summarize, software companies can benefit from creating a software ecosystem based on shared Open Source, including partners, customers, end users and sometimes competitors. The latter is not too far-fetched as many competitors may be entangled in the same Open Source development, being R&D partners in an industry-wide collaboration. Such collaboration significantly raises the market entrance bar for competitors with proprietary offerings – who would have costlier development and maintenance as well as the burden of alone having to prove its value on the market.


Accelerate growth of business
Disrupt market logic
Open up for alternative business opportunities
Only product value enhanced by Open Source
Difficult to control Open Source evolution – passive receivers
Collaborative, though mainly passive receivers
Basic Open Source directives and policies are established
Contribution strategy formulated
Proprietary strategic technology
Able to create and direct ecosystem
Directed by business aspects of Open Source
Leadership that coaches and support
Participatory culture
Authority on Open Source
Crowd-based requirements engineering
Industry-wide collaborations
Control ecosystems
Strategic technology opened up as market disruptor
Open Source contribution strategy as product strategy
Open Source driven product innovation
New markets, products, services and business models due to Open Source
Increased sales and profitability
Value extracted from own ecosystem
High flexibility in capture multiple revenue streams
Faster growth than the competition

Get inspired

This scenario has been based on case studies of different companies that have made this journey, to scale with Open Source. Learn from their experiences, what they gained and what they had to overcome.

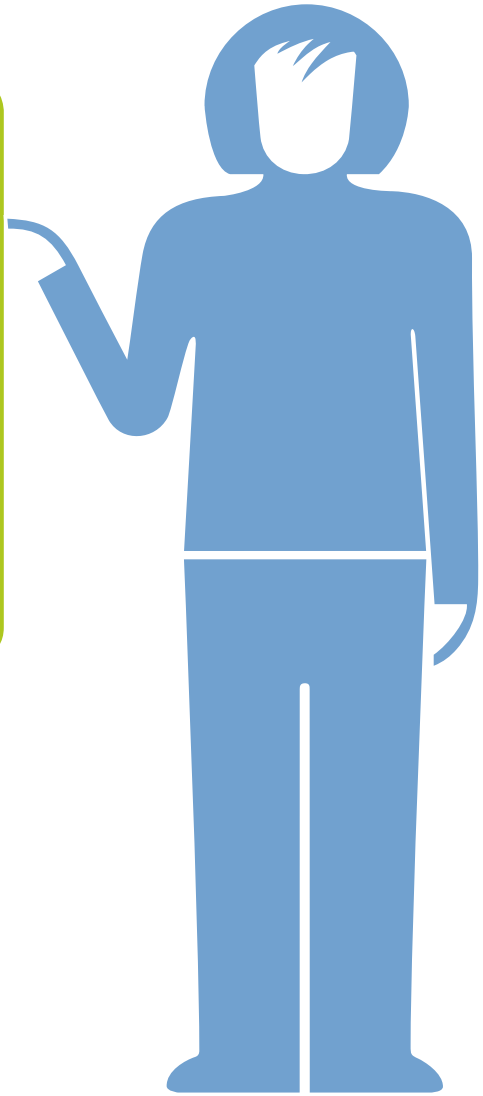


Pushing the boundaries

There are companies that relies on cloud technology infrastructure more than others. If they also happen to include gigantic data transfers in their offerings, it's not unlikely that they will have a very close look into the ecosystem Netflix have created. Read about the company that employs more than 700 engineers that more or less gives away everything they create just to keep the business flourish.

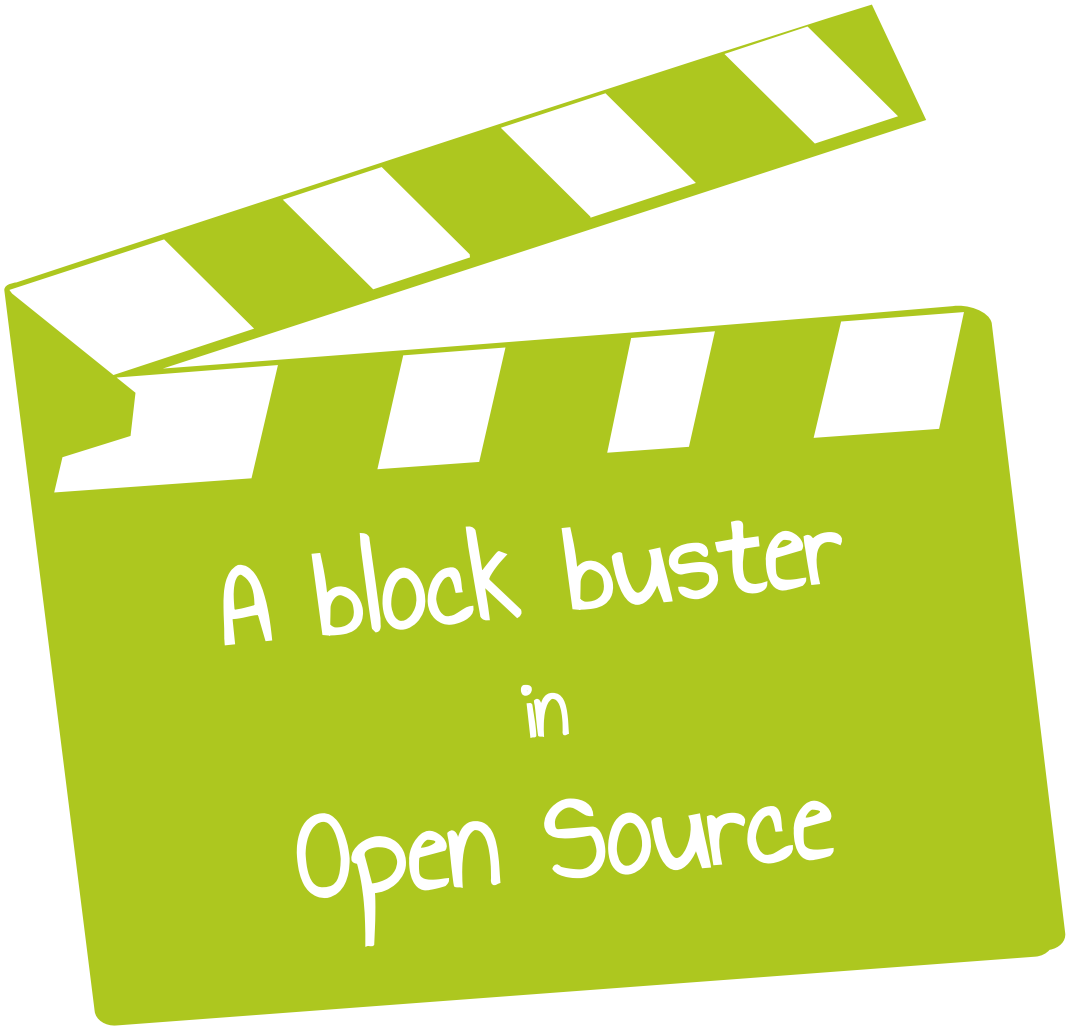
Servitization

Open source and ecosystems are almost always the preferred way when scaling a business from products into services. Take again Netflix, who provided a DVD-by-mail service before creating their video streaming service. They likely made a transformation such as the one the Servitization scenario describes. It's the next chapter in this book, "Add supplementary services." Read it as well!



www.dbooks.org

Pushing the boundaries



For the last years, Netflix has accounted for more than a third of the Internet traffic streaming into North American homes every night. That's more than the combined traffic of the trailing contenders such as YouTube, Hulu, Amazon, and iTunes.

At any moment, Netflix draws upon 10.000 to 20.000 servers running in Amazon data centers. The computers handle customer information, video recommendations, digital rights management, encoding of video files into different formats and monitoring the performance of the systems. When a new device as for instance an upgraded game console or a smartphone comes along, Netflix uses thousands of extra servers to reformat movie files and to serve the new users. Netflix has a renowned reputation for pushing cloud computing and Amazon Web Services to its limits.

From the very start, Netflix has been forced to build much of the software from ground up. Since they rely on Amazon data centers, their 700+ engineers focus on tools that for instance automate the way in which thousands of cloud servers are started and configured. Like Google, Facebook, LinkedIn and Twitter, Netflix depends largely on Open Source for much of the software that underlies their operations. The companies compete in fact on who's paying most for the most clever engineers, to give away their outcome so that others can build on top of it.

Giving away their technical wizardry for free and to rely on 4.000+ volunteer programmers makes perfect sense to Netflix. At the end of the day, they will gain from more robust code that has been infused with bleeding-edge innovation.

Together, these companies forge the cutting-edge Open Source cloud computing technology that mainstream companies will use in the years to come. It is essential to Netflix to have a leading role in streaming video cloud technology to maintain their forerunner market position. They have to accept that their competitors – old ones working with cable technology as well as new ones working with the web – in a flick can become their partners.

To ensure the case, Netflix has created a great arsenal of open-source cloud computing technology. They have tools to install preset packages of software on Amazon servers, to reconfigure them and to test them without causing any downtime. The so-called Simian Army, another set

www.dbooks.org

of applications, purposefully try to wreak havoc on their infrastructure in a bid to find holes and performance problems. Chaos Monkey, for instance, simulates small outages by randomly turning services off, while Chaos Kong takes down an entire data center.

Eventually, Netflix wants to use an open source platform. Instead of releasing cool but disparate projects, the company wants to put together a cloud management system that software developers can poke and prod and advance. A platform would provide an opportunity to widen the offering beyond the core business.

Interestingly enough, Netflix's Open Source strategy is not of any particular old age. It all started in 2011. Only a few years later, they had built a bustling community and ecosystem. Today, Netflix is crystal clear on their corporate view on Open Source. It is not only a mean for development – it's a strategic weapon for their business. We should expect the "More to come, stay tuned!" screen on their business outlook.



Sources and more reading

Netflix, Reed Hastings Survive Missteps to Join Silicon Valley's Elite, 2013, www.businessweek.com

Netflix Announces $100,000 in Prizes for Coders, 2013, www.businessweek.com

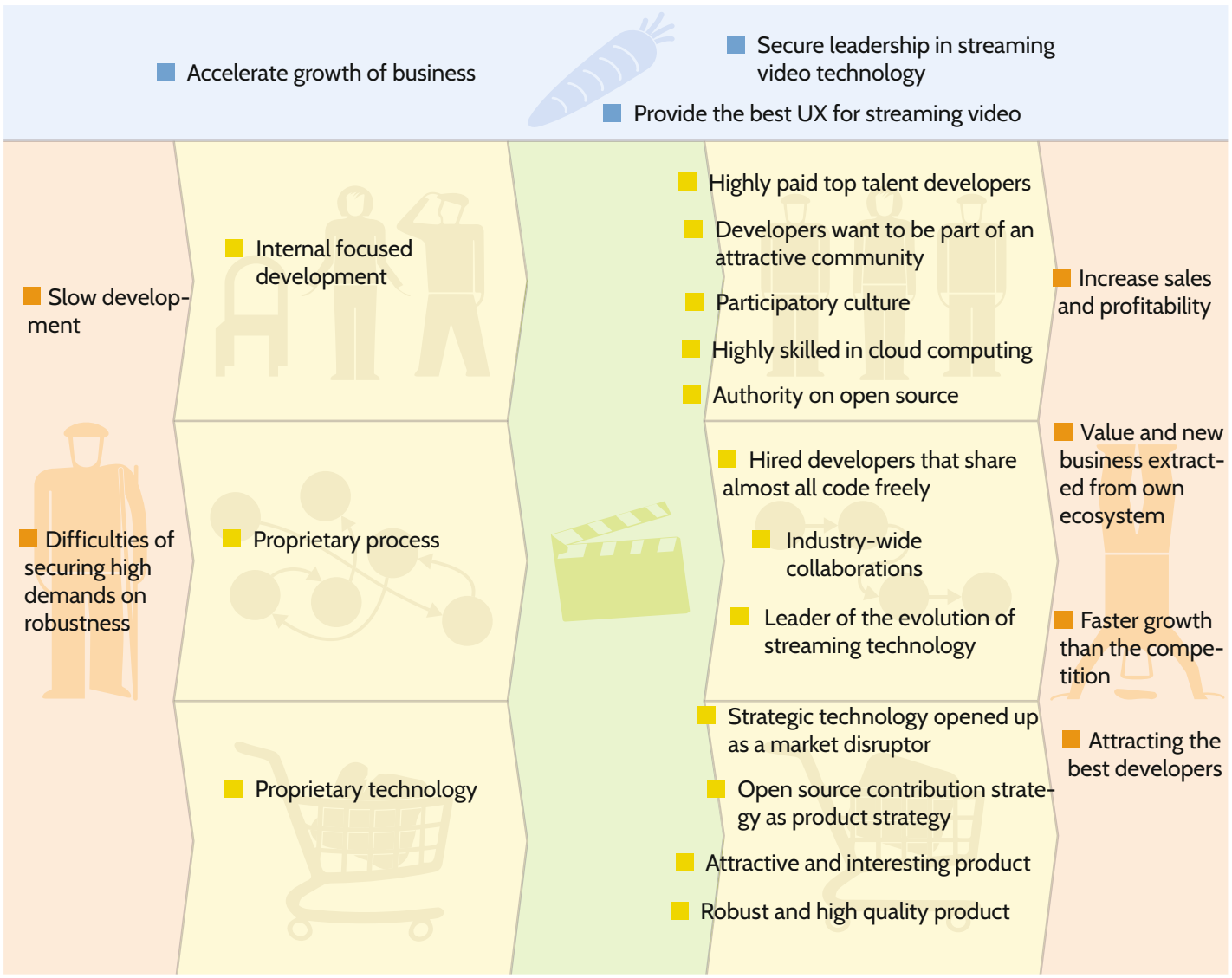
Netflix and YouTube Dominate Online Video. Can Amazon Catch Up?, 2013, www.businessweek.com

NetflixOSS Meetup, 2013, www.slideshare.net

techblog.netflix.com

Netflix Open Source software downloads

netflix.github.io/#repo



www.dbooks.org

Add supplementary services



> A service is the means of delivering value to customers by facilitating outcomes customers want to achieve without the ownership of specific costs and risks.

ITIL* defines service this way. It basically means that the customer gets something they want without having to bother about the supplier's efforts to provide it.

Services are stand-alone offerings, but could also be offered with a product as a supplement.

To get to the grips with servitization, see also:

- **Made to Serve:** How Manufacturers can Compete Through Servitization and product Service Systems, Timothy Baines
- **The Startup Owner's Manual:** The Step-By-Step Guide for Building a Great Company, Steve Blank

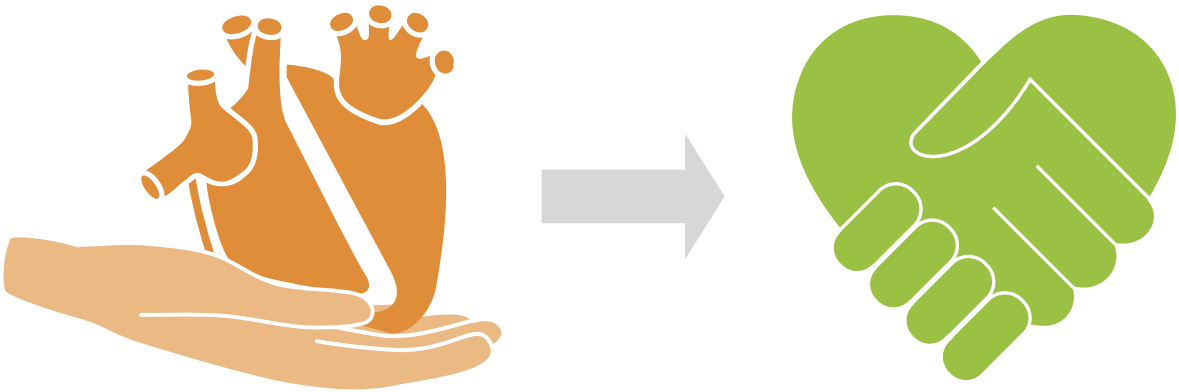


* ITIL 2011, (Information Technology Structure Library)

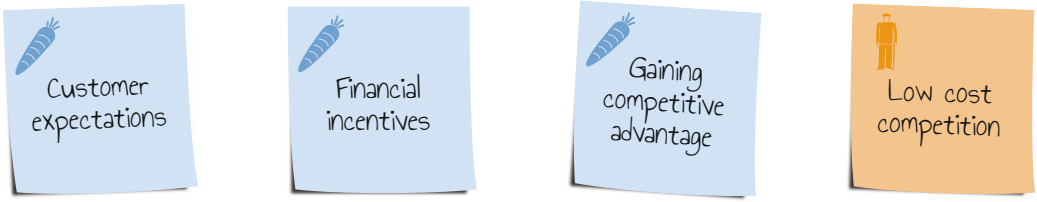
www.dbooks.org

servitization

[ser · vi · ti · sa · tion]



In essence servitization is a transformation journey - it involves companies developing the capabilities needed to provide services and solutions that either supplement or replace their traditional product offerings. Recent technological advances such as cloud computing, big data analytics, mobility and social media have enabled the servitization trend.



Servitization is used when a company introduces a service offering as a means to further satisfy their customers' needs, to enhance profitability, to gain a competitive advantage or to avoid competing with low-cost countries on a cost alone basis. The theory suggests there are three levels of product-to-service transformations, but there are no clearly delineated boundaries. It's rather a matter of how your value proposition is defined.



To completely replace a product with a service requires a long-term investment.

In this case there is no product or owner. The service is consumed at the same time as it's delivered.

Companies that made this journey can for instance be found in the entertainment industry.

Example: Netflix



A middle-road approach is to keep the product and transform the business model to be subscription-based.

The owner of the product is now the service provider. Companies that made this journey can be found where buying the product requires a substantial investment.

Examples: Rolls-Royce with its Power-by-the-hour aircraft engine program and Xerox printer service.



Least complex is to keep the product and purchase-based business model, but add supplementary services such as maintenance and premium programs.

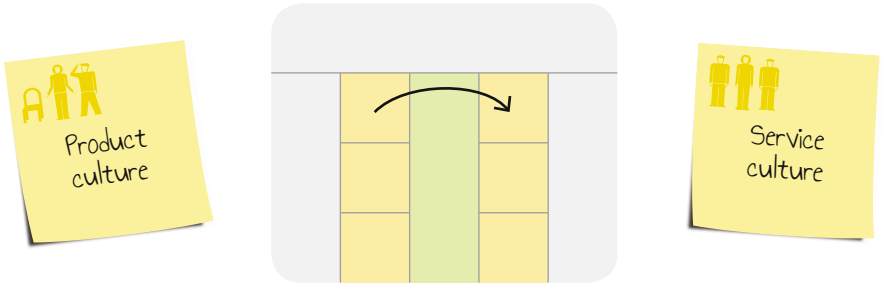
The owner of the product is in this case still the customer. Numerous businesses successfully adopts this model.

Examples: Ericsson offers maintenance of their telecommunication equipment.

www.dbooks.org

Turning a legacy product business into a cloud-based service business has become the Holy Grail of the software industry in recent years. Being able to provide customer value by making services that for instance use data from the billions of Internet of Things sensors, turns out to be a money-maker. It's not that easy, of course. You have to make sense of the data, turn it into knowledge and meaningful actions. Parking place sensors are practically useless if you can't come up with a clever way to help your customers quickly find a free parking place. There are numerous examples of companies that have successfully made the journey to replace their products with services and by that increased their profit. Yet, it's far from a done-deal, whereas many also fail. Most casualties are usually claimed during the transformation phase. Learning from the pitfalls and how successful companies made it, significantly increase a company's chance to succeed. The solution described in this chapter has proven to be fruitful.

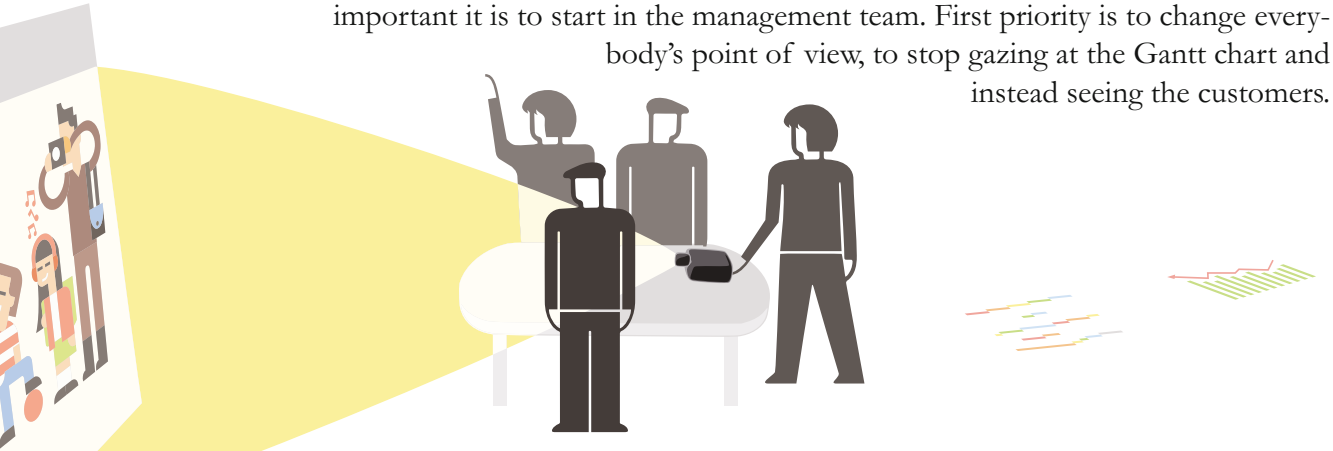




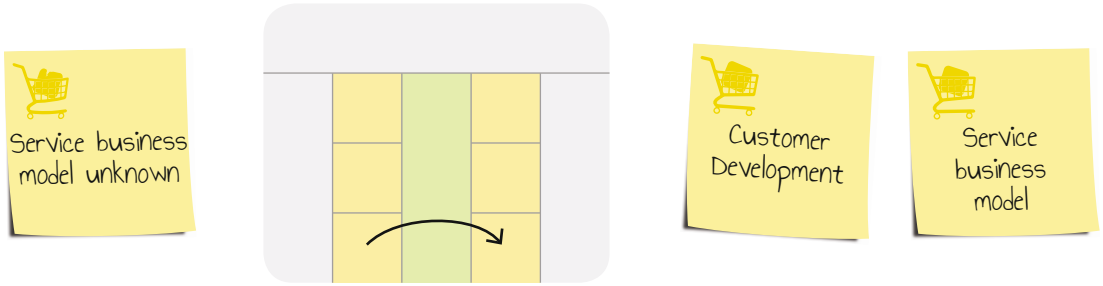
Consider for a minute the dynamics within the management group. Who gets to talk most on the management group meetings? This will change. Substantial amount of time and resources that are currently allocated to track and report development and manufacturing will decrease to an absolute minimum. Sales, or rather the new function that sales will have to become, will get the most attention. Expect clashes since not everybody will be able to free themselves from their previous roles and traditional ways to run the business.

Being able to offer a service is a huge opportunity that can completely change your business model. Don't think too long about it. Setting up a profitable service takes time and it's not unlikely that your competitors have already started. Agility will turn out being your key ability.

Regardless where you aim in your scaling effort, it cannot be emphasized enough how important it is to start in the management team. First priority is to change everybody's point of view, to stop gazing at the Gantt chart and instead seeing the customers.



Another priority, and indeed a great challenge, is to develop a scalable, repeatable and profitable business model. In terms of creating it, there isn't really any difference between an established product business that wants to start a service and a start-up business. They have an idea of a service business but the value proposition hasn't been carved out entirely. Both have to ask and challenge the same questions.

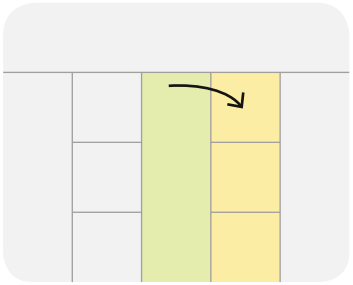


To support in finding the answers, many successful, web-based companies – including Spotify, Dropbox, Airbnb and IMVU – use the customer development method described in the book "The Lean Startup".* It's a very good start, no matter from where you come.

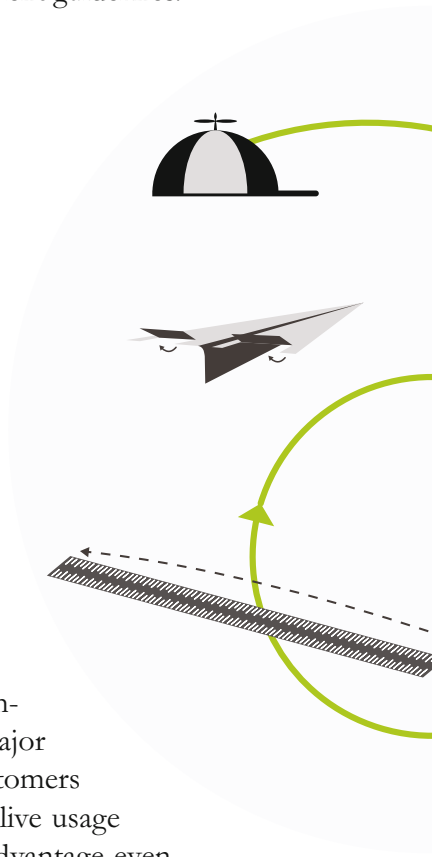
* The Lean Startup. Crown Publishing Group. Eric Ries

www.dbooks.org

Customers have to get value day after day, month after month, to stay as such. They have very high demands on availability. When at least 99.9% availability is expected on a normal service, there's absolutely no room for temporary glitches. The ITIL 2011 has shown to be a great source of knowledge to support how to get there. Study in particular the Service Management guidelines.



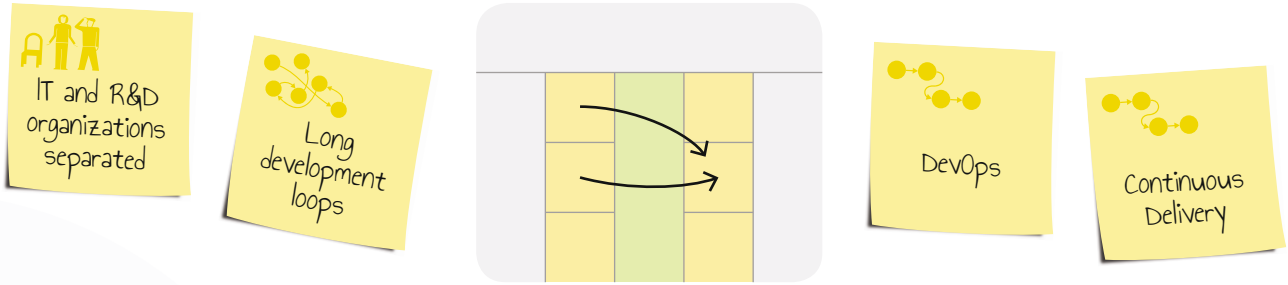


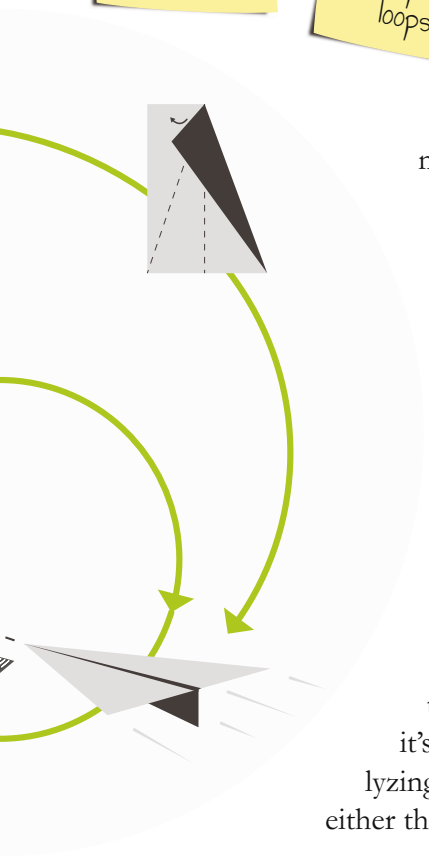


How do you provide a service that customers willingly pay for month after month? You have to get to know them and demonstrate that your service adds value to every one of them, every day. The technology gets secondary – the customer experience becomes primary. What are their needs? Understanding your customers and their behavior will be key. How do they use your service? Fortunately, it's much easier to monitor customer behavior when they're using services than it is when they're using products.

A huge advantage with software services is that you can prototype improvements and new features in a small scale without having to make major investments. The service can grow as the business grows and your customers can benefit from new functionality instantly. Further, being able to use live usage data and prioritize development that improves the service, makes this advantage even more valuable.

The development work lends itself to be carried out in an iterative way, to create a so-called





minimum viable product every week, or even every day. There are Agile development techniques such as Continuous Delivery, A/B testing and DevOps* that perfectly support this way of working. An iteration would only comprise a few of the highest ranked work items in a priority-ordered list, called the backlog.

After having provided the new features to your customers, you can measure, analyze and improve them. If a feature doesn't live up to the customer's expectations, the design has to be improved. New features are not taken away from the developer's backlog until the customers show, either through surveys or logged behavior, that everything works nicely.

Simply put – fail fast, learn fast, and improve fast.

This is a never ending cycle of making a hypothesis about a new customer feature, building the minimum viable product, looking into how it's used, learning from it, making a new hypothesis, building, releasing, analyzing and learning, and on and on. There will always be high time to improve either the business model or the service.

Experiences have shown that the companies that have created small, autonomous teams that work as start-ups are the ones that also are the most successful. Many of them work according to the customer development methods described in "The Lean Startup".

* A practice that emphasizes the collaboration and communication of software developers and IT professionals

www.dbooks.org



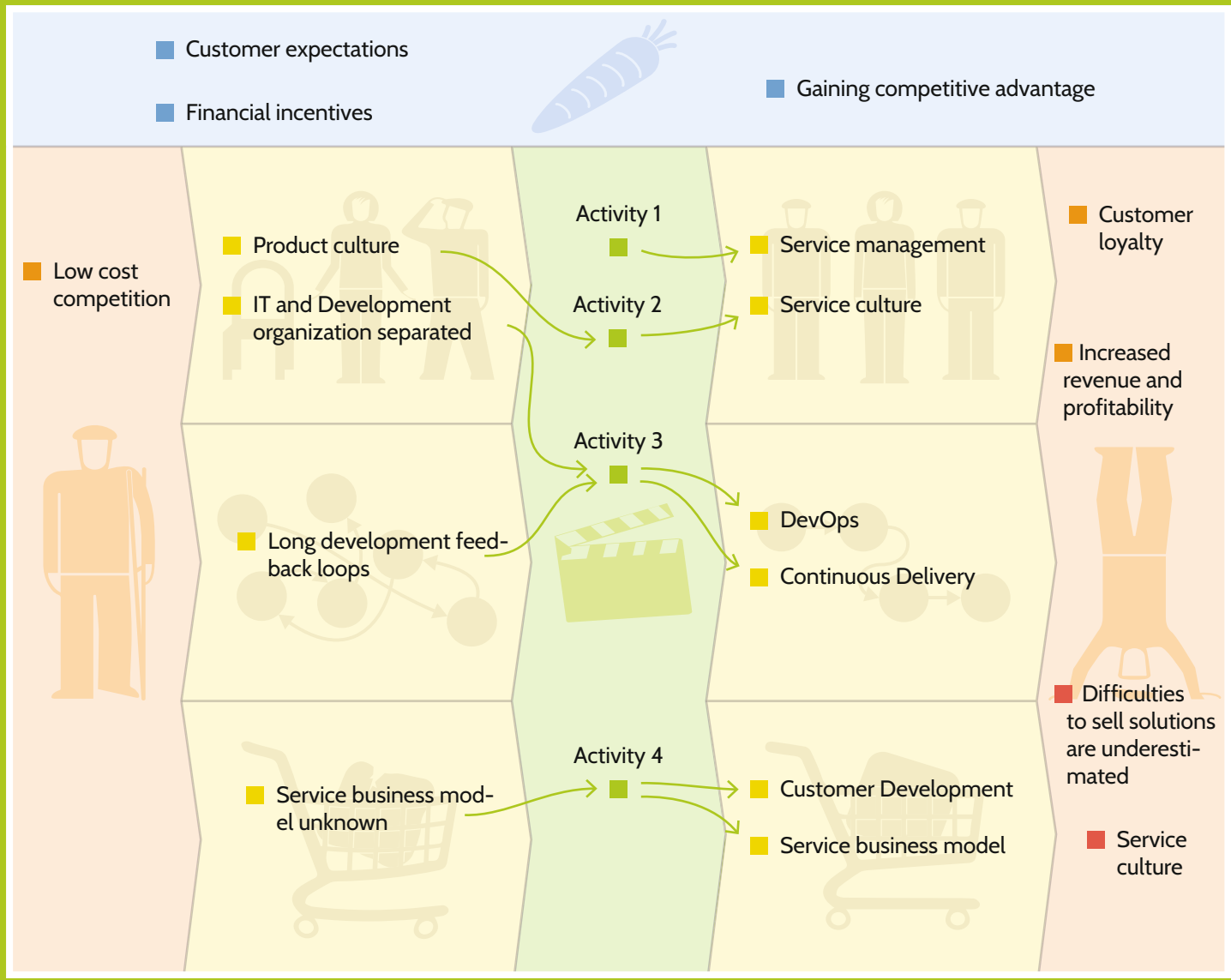
The challenges are plenty fold, as can be expected. While many companies are successful, many also have difficulties in seeing the benefits. Most commonly, they experience problems with:

- Conflicts in the organization
- Difficulties to sell solutions that aren't tangible
- Creating a service oriented business culture

The pit holes along any servitization journey are numerous and failing to get around them will inevitably jeopardize the transformation process. But, there's no reason to be discouraged. Despite a massive change in business and ways of working, the servitization journey is proven to be truly worthwhile. Research shows, for instance, that successful service providers clearly gain:

- Customer loyalty
- Increased revenue and profitability
- Robust cash flow and revenue streams

The servitization canvas helps to avoid making old mistakes in getting these gains.



www.dbooks.org

Get insights

This scenario has been based on case studies of different companies that have made this journey, to scale with Servitization. Learn from their experiences, what they gained and what they had to overcome.



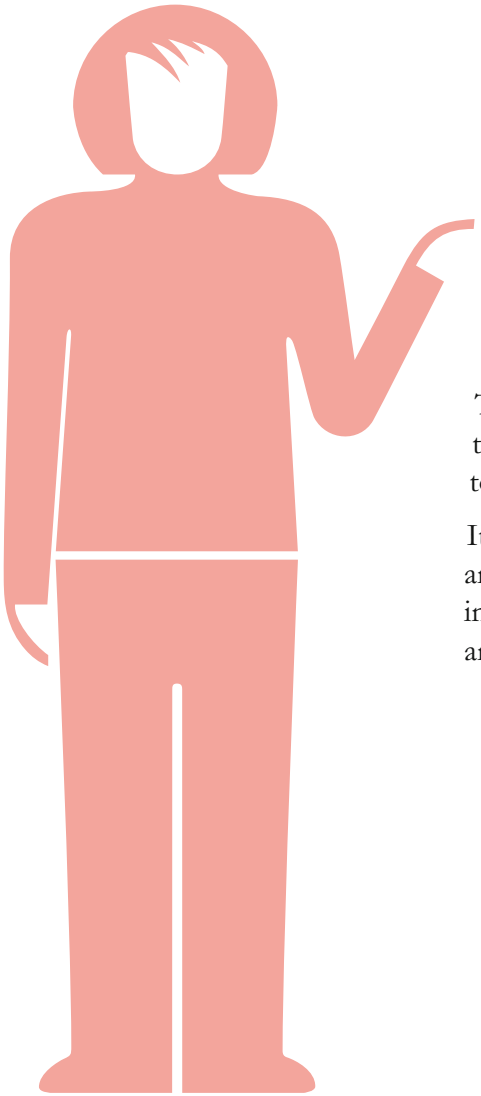
Adding Internet to things

Read about the company that boosted their B2B sales by spicing their lawnmowers products with IoT.



Boosting product sales by services

Learn from the company who aimed too high with a worldwide download service for the man on the street.



One more thing

When scaling a software organization through servitization, the software architecture has to be adapted.

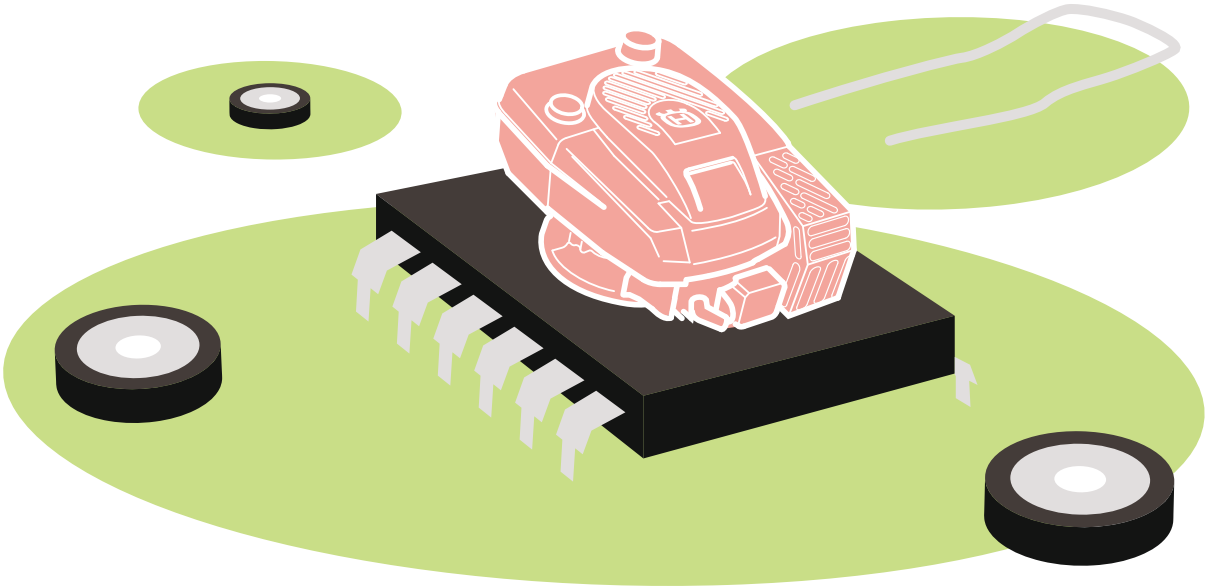
The chapter **First things first** describes a canvas that was created for this purpose, as a guide on how to improve the software's internal structure.

It is furthermore strongly recommended to work in an Agile way. Find canvases for Agile development in the chapters **Pump up the volume**, **Deliver 24/7** and **Agile and Disciplined**.

www.dbooks.org

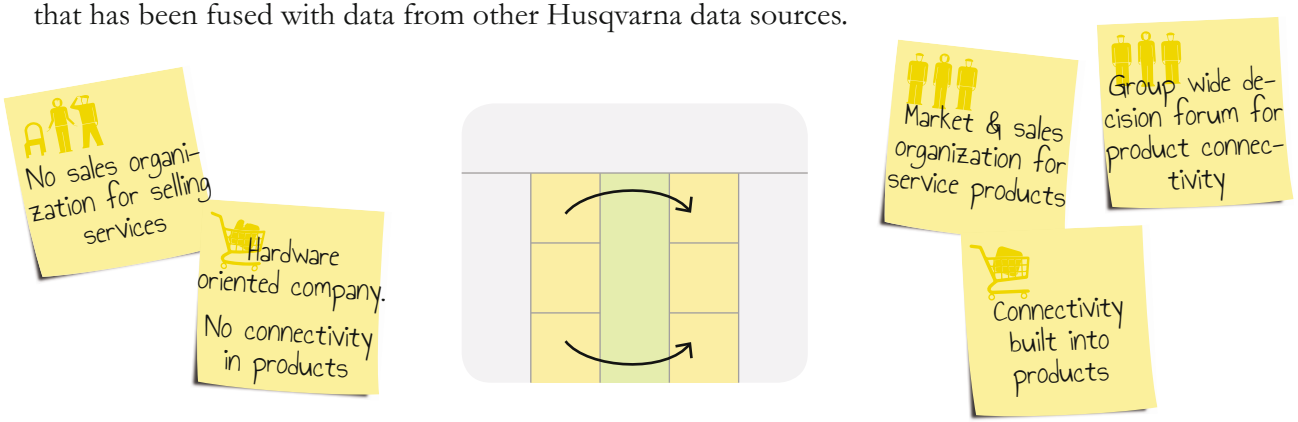
CASE STUDY / Add supplementary services

Adding Internet to things



Husqvarna is a global manufacturer of lawn and garden equipment, offering products varying from chainsaws to lawn mowers. They are now in a shift from offering mechanical products to also produce products having electronics and software. An increasing amount of unique product features now builds on software. The shift started in the mid 1990's when the first robotic lawn mowers were launched, but until 2016 the products had virtually no connectivity. It's connectivity that makes the machines able to communicate and possible to integrate with customer services. Husqvarna Fleet Services is such a service.

The hardware of the system consists of a sensor that is mounted on the machine, an operator tag that is carried by the person who uses the machine and a base station that is placed in the customer's garage. The sensor collects data about how the machine is used out on the field. It also keeps track of who is operating the machine. When the machine is back in the garage, the data is collected and sent through a gateway to Husqvarna Fleet Services cloud data services. Operators, managers and technicians at the companies that subscribe to the service can then get information such as vibration levels, service needs and machine usage based on collected data that has been fused with data from other Husqvarna data sources.

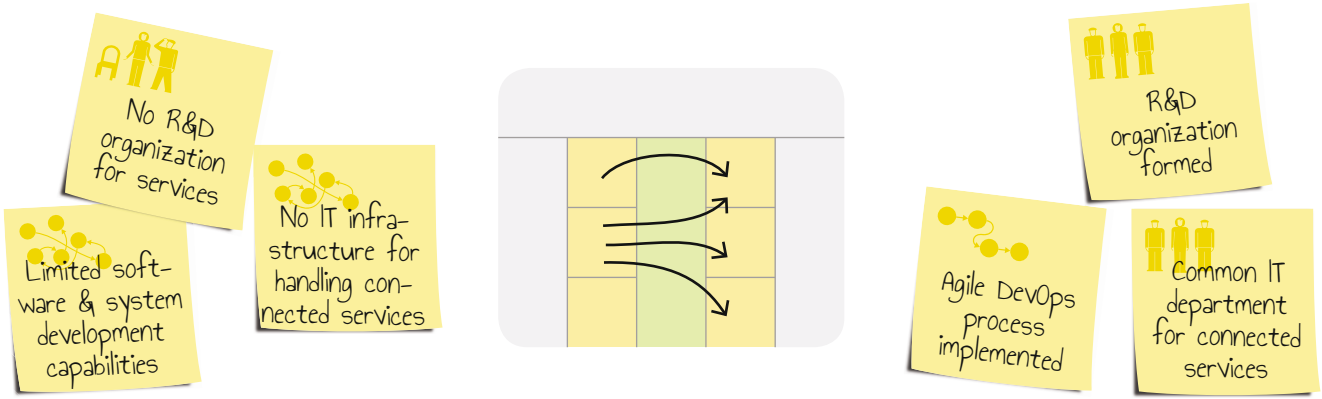


Husqvarna Fleet Services started in 2007 as an idea in the After-sales department, basically to offer a service that help Commercial Lawn and Garden customers managing their fleet of Husqvarna products. Until 2011, all development was run by After-sales as a study on a very limited budget. To create the proof-of-concept, they cooperated with an external technology company. From that point, in 2011, the project gained increasing support from the R&D department. In August 2014, a beta version of the service was finally released to a limited set of customers. A new R&D department was started in December, with the responsibility to host all Husqvarna service products, one of them being the Husqvarna Fleet Services. A group wide decision forum for product connectivity and a department for marketing and sales was as well created. The first limited commercial release of Husqvarna Fleet Services was released mid-2016.

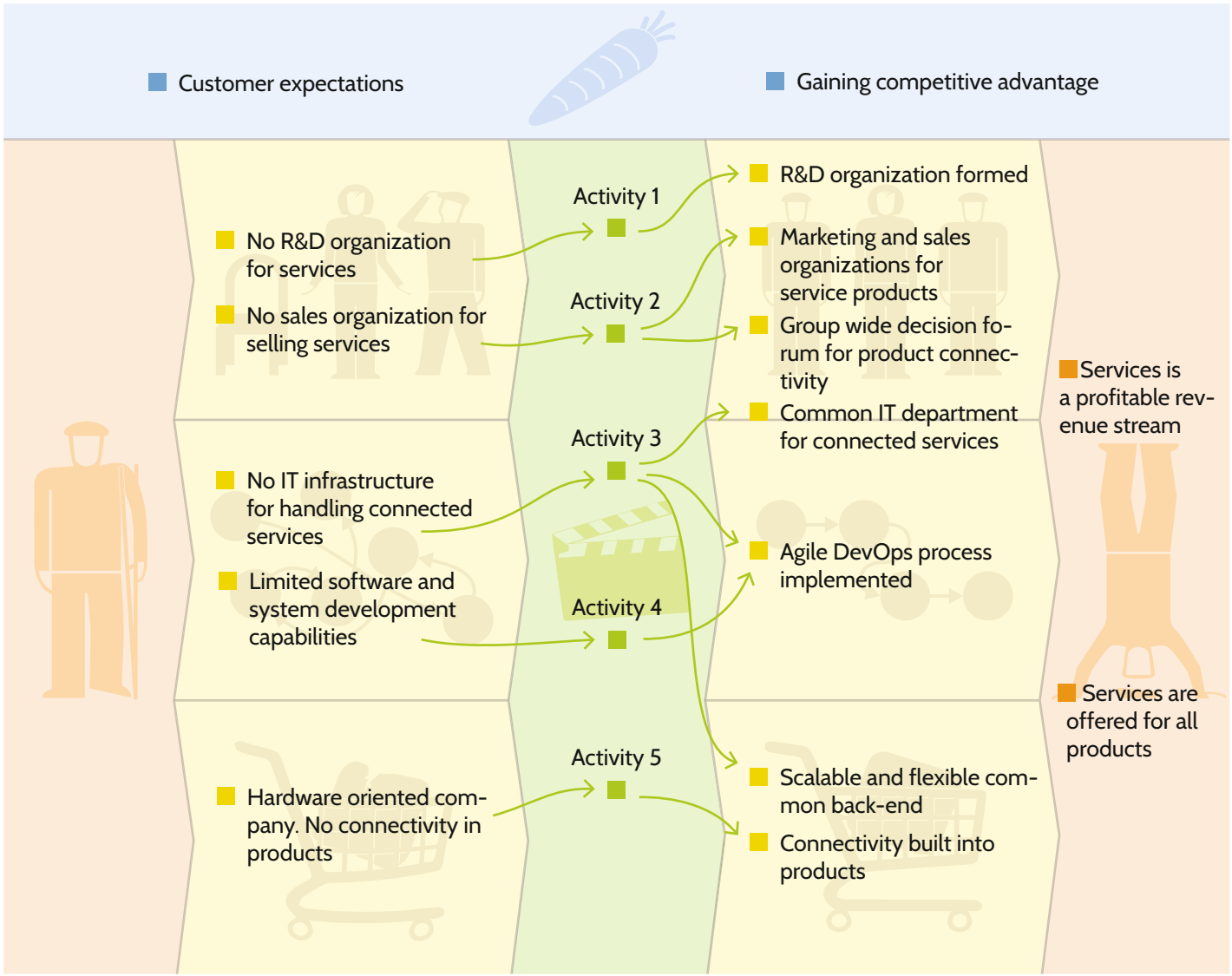


www.dbooks.org

Husqvarna's products were becoming smart devices – but this wasn't enough: Husqvarna's organization had to become smarter, too. As a hardware-oriented company, Husqvarna had initially very limited software and system development capabilities. An important step was to form an R&D department that could develop new services. The IT organization had to create a new department as well, with responsibility to build and operate the IT infrastructure needed for the service. They worked independently from the original parts of the IT department. The IT organization had to work in a Bi-modal way, so to say in one speed for the original IT work and in another speed for the rapidly moving services developed by the DevOps teams.



Husqvarna saw the demand from the market, the need for a product like Husqvarna Fleet Service. They identified a market trend such as digitalization and Internet of Things and now the organization is changing accordingly. What originally started as an experiment is now reshaping the identity of the entire company – it's a transformation from a traditional, product-oriented manufacturing business to a business where services lead the way towards the future. It has taken a long time to get where they are now, about nine years. But the idea of introducing services was a good one, an idea that eventually made the whole company line up and work towards a common goal. As this is written Husqvarna is still on the journey, but its chances to succeed in its servitization is considered very good.



www.dbooks.org

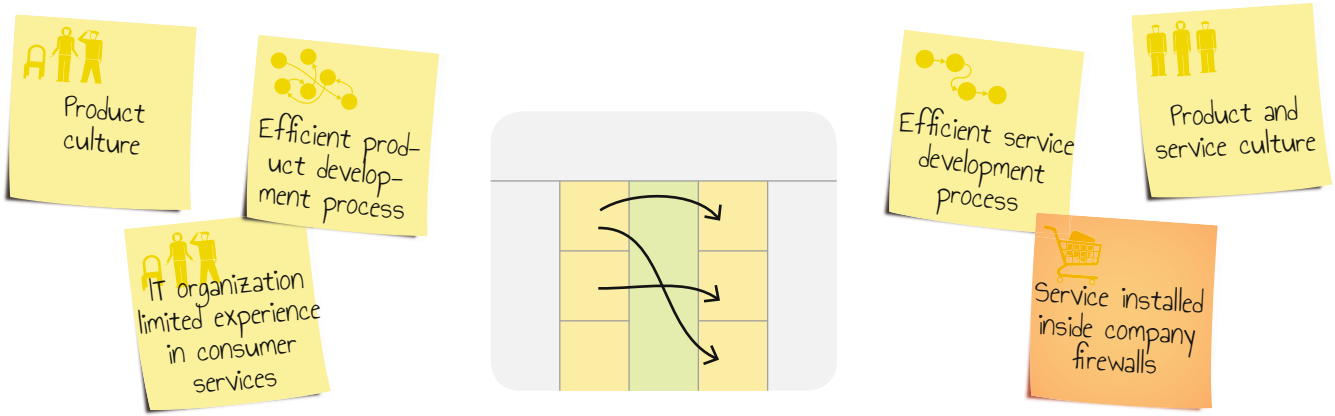
CASE STUDY / Add supplementary services

Boosting product sales by services



This case study is about how the mobile phone manufacturer Sony Ericsson went from only offering physical products to also include a music listening service called "PlayNow plus". Sony Ericsson released their first Walkman phone in 2005. It was the world's first phone aimed for listening to music. The PlayNow plus service was basically introduced to increase sales of mobile phones, but also to be a step to keep the lead in the global music listening business. The service didn't have to be profitable on its own.

The case study confirms that the culture, the understanding of how to sell the service, and how new ways of working are defined are the three most common challenges when introducing services. Atos Consulting* also confirmed these findings. The transformation requires several steps, including adjustment of KPIs, redesigning processes, aligning management, organization (not least the IT department), people, and culture. It is virtually impossible to shift the entire organization at once. The servitization transformation is a journey and not a one-time change.



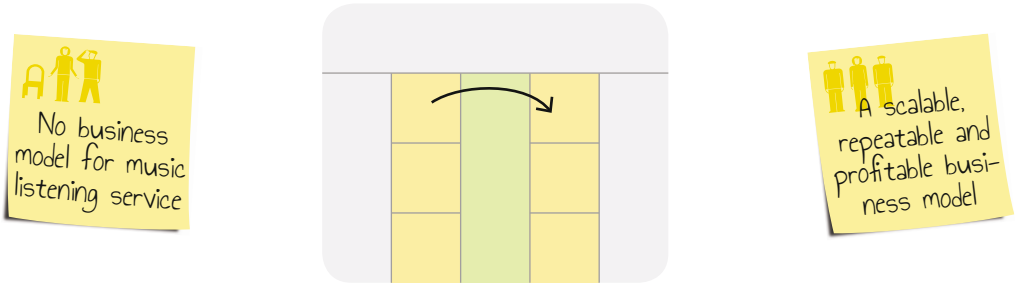
Sony Ericsson was a product company with very little experience in delivering services. The service organization experienced huge difficulties in both changing the culture of Sony Ericsson as educating personnel in what it meant to deliver a service. Part of the explanation is that almost all sales came from selling mobile phones; no profit was expected from the PlayNow plus. Sony Ericsson continued to focus on product development and also expected the PlayNow team to align with their long-lasting product development cycles. With customers expecting new functionality continuously, the way the service developed and how it was operated was far from optimal.

The IT organization did not have any prior experience of external customer facing applications. Due to security reasons, the service had to be hosted behind the company firewalls. The IT organization simply didn't understand the implications of this decision. This led to enormous problems,

* Atos 2011, www.consultancy.nl

www.dbooks.org

such as having to shut down the service during weekends during the regular maintenance of the internal IT systems. A service like PlayNow plus must be available 99.9% of the time, if its users should even think about using the service as their main way to listen to music.



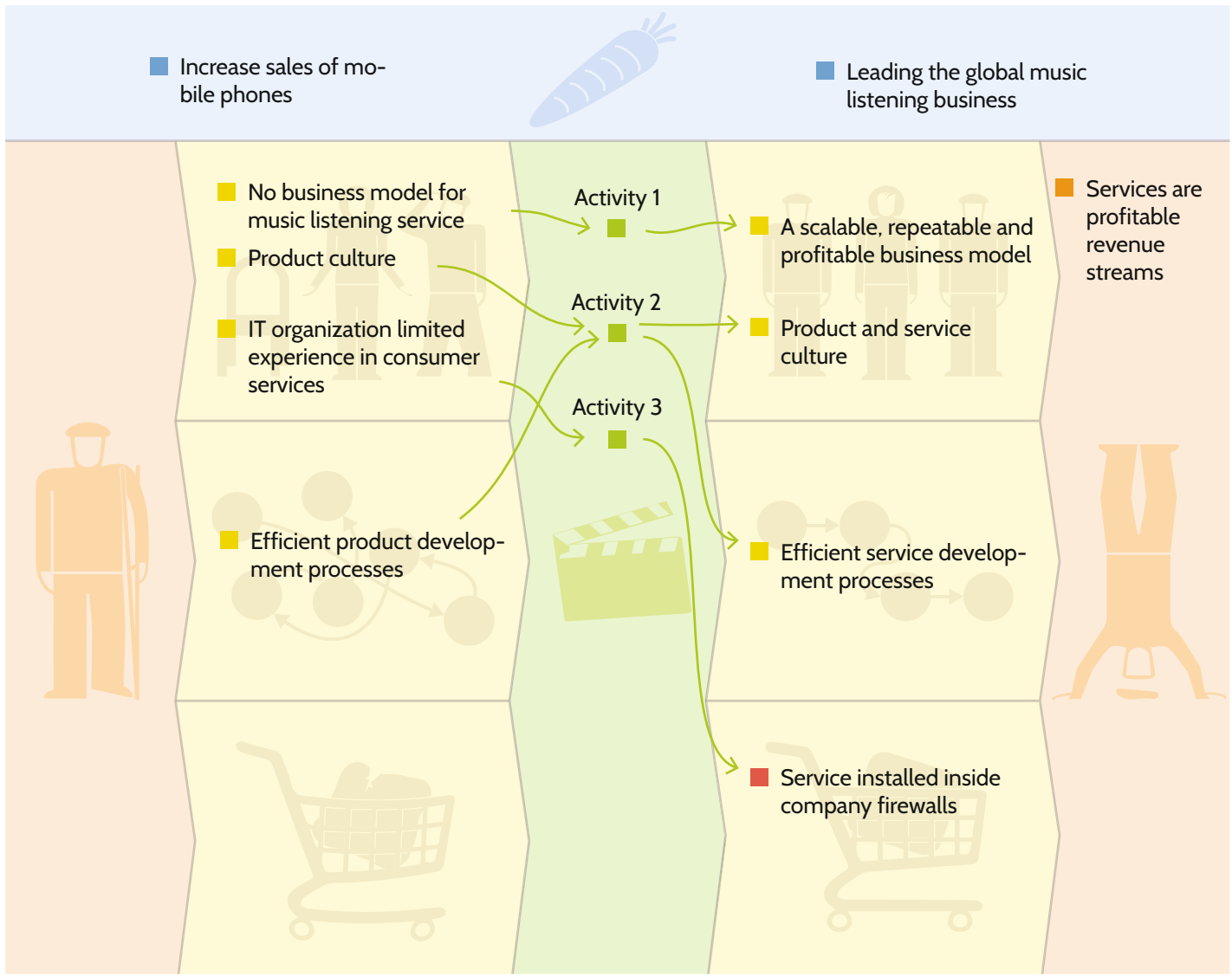
While cultural problems were among of the biggest internal hurdles to overcome, communicating the service business model to potential customers became their biggest external problem. The service organization lacked basic knowledge about the market. Two questions they didn't ask were:

- Was there a need for the service or did the users want another kind of music service?
- Did the users like the service or was there a need for updates and new functionality?

The PlayNow plus service was launched to the Nordic countries in August 2008. As a reference, Spotify was launched shortly after, in October. PlayNow plus offered DRM free (without digital rights management) MP3 tracks, while Spotify offered a streaming service. They offered basically the same service, but in two completely different ways.

Spotify reported a loss of $4.4 million in 2008. But, as it had a long-term goal to be profitable, it could take the loss for an extended amount of time. PlayNow plus was never considered to be a future profit maker, nor was it seen as an investment. The service was terminated in 2010, as a decision to cut costs.

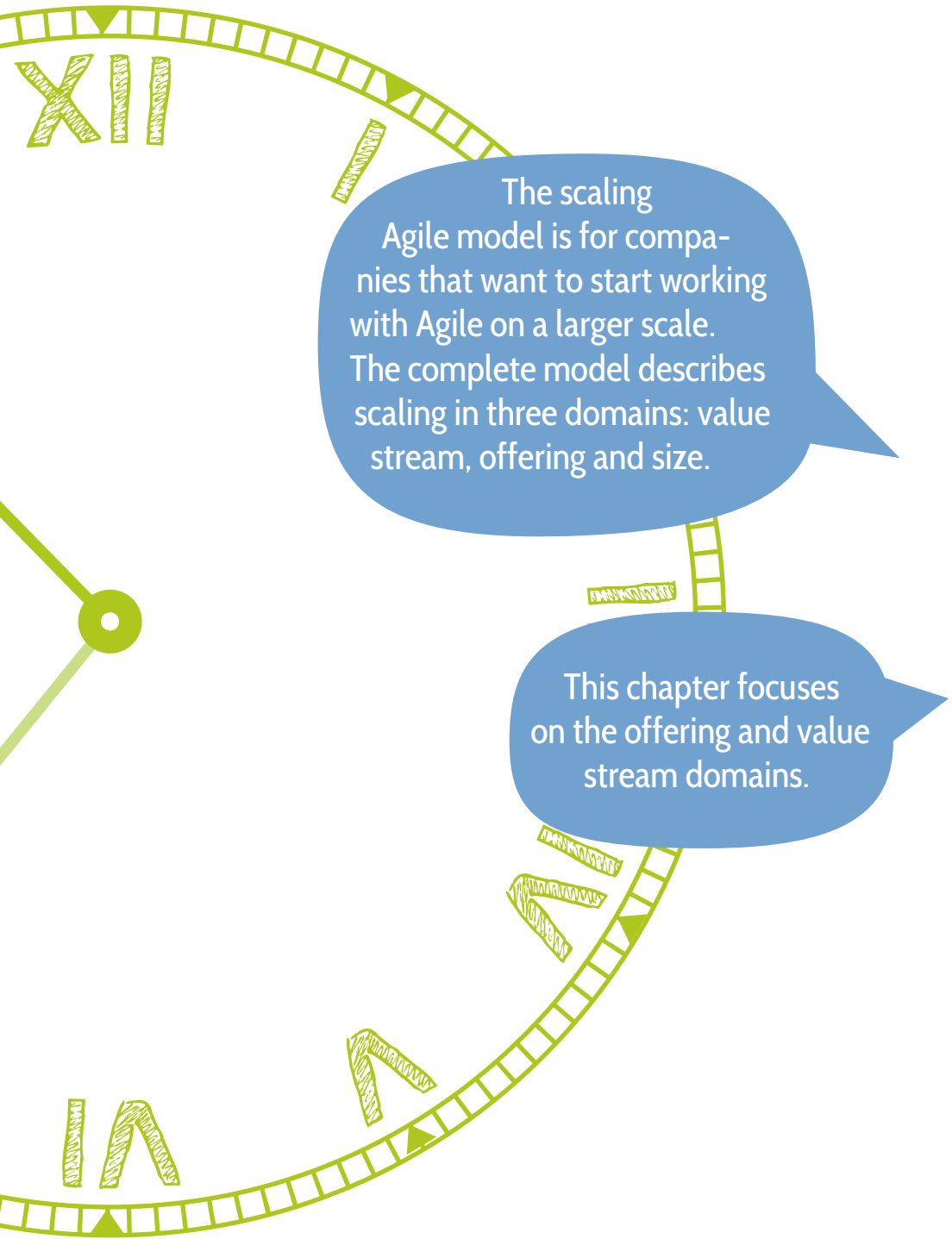




www.dbooks.org

Deliver 24/7







www.dbooks.org

About Agile

During the 1990s, a number of lightweight software development methods evolved in reaction to the prevailing waterfall software development methods. They all follow the principles that were outlined in the Agile manifesto 2001.* There are many agile development methods now. Most of them promote teamwork, collaboration, and process adaptability throughout the product development life cycle.

Agile development methods break product development work into small increments allowing frequent feedback on value and quality.

Scaling Agile is a concept that helps you spread and consolidate the agile philosophy throughout the organization to follow the Agile Manifesto that says that "our highest priority is to satisfy the customer through early and continuous delivery of valuable software."

This type of change is often very thorough and includes a review of leadership, governance and decision-making structures as well as the changing roles and modified organization. The choice depends on the company's drivers – value creation, efficiency, innovation, or something else.

* http://agilemanifesto.org/

Companies that succeed with the scaling Agile process are characterized by three things:



Co-located, cross-functional teams

The teams contain all the necessary competence to deliver value, for instance developers, testers, architects and business representatives. The teams are as well co-located to get efficient face-to-face communication.



Delegative style leadership

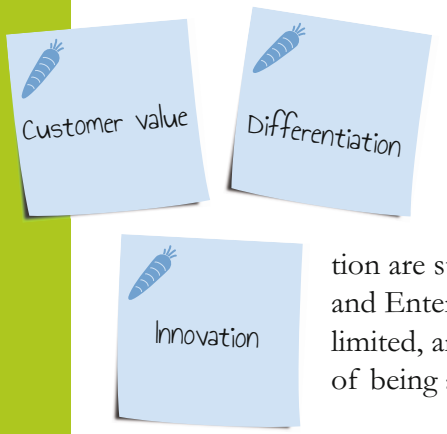
Management must understand the agile values and make it necessary to implement the planned change. Agile methods also requires that the leadership focus on teams in a very different way than the traditional command and control style. A more delegative and cooperative leadership style is needed.



Iterative approach with short feedback loops

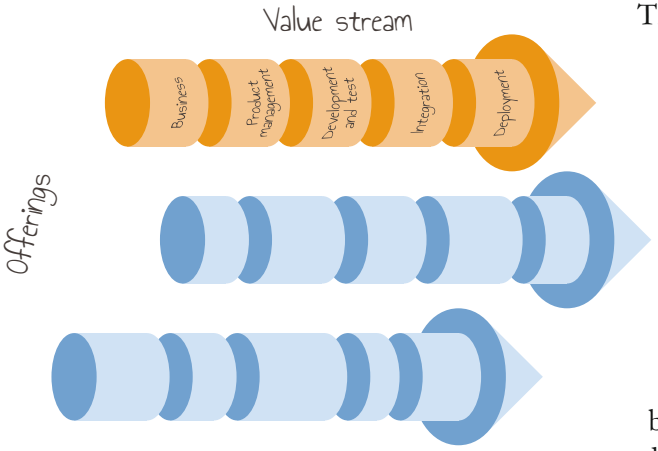
Technology and infrastructure will enable fast feedback and fast, frequent release cycles. At the end of each iteration, stakeholders as well as customers review the progress and re-evaluate priorities to optimize the return on investment and to ensure alignment with customer needs.

www.dbooks.org



Scaling agile

Many large companies struggle with timely delivery of products and services that customers really want. While agile methods are widely adopted, a key challenge is to scale the agile approach to the corporate level. While software development teams may follow agile methods such as Scrum, customers may not benefit if other key parts of the organization are still operating based on a waterfall philosophy. Adoption of Lean Thinking and Enterprise Agile philosophies that focus on end-to-end systems thinking is still limited, and the software industry has a long way to go to achieve the true benefits of being agile.



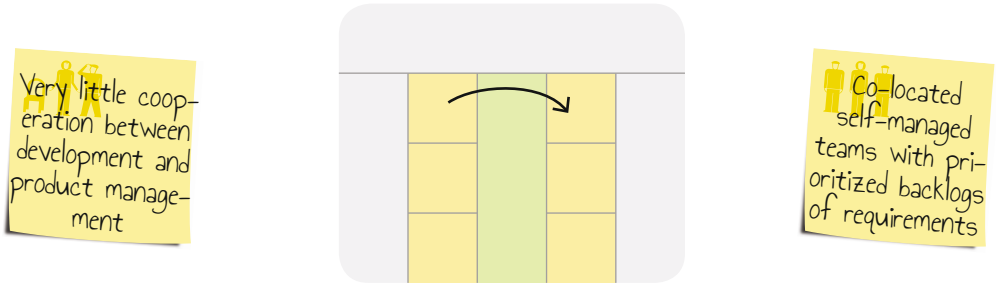
There are three reasons why a company would want to scale up agile to the whole organization:

1. Doing the right thing at the right time,
2. Focusing on delivering value to the customer, and
3. Improving the capability to innovate.

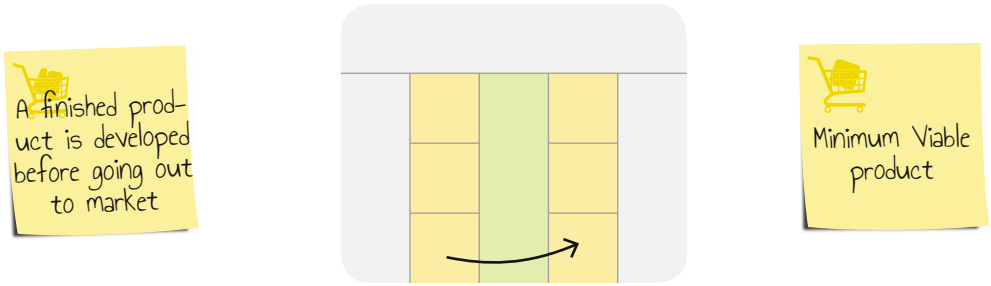
Everything is sprung out of the need to become stronger, to innovate better products and to better differentiate between products.

The scaling objective (what we want to become) is in the theory called Scaled Agile. Scaling Agile is the transformation that gets us there, towards the Scaled Agile. There is no actual end state, since the market, technical trends and so on vary over time.

In this scenario we deal with two levels of scaling objectives. The primary objective is to scale the value stream – this is what being Agile is all about, to take full responsibility from idea to payment, i.e. to have a working end-to-end flow within the organization. The secondary objective is to scale the offering – this is to scale stand-alone products or services. Each offering will have its own value stream and perform as a "mini-company" in its own.



It's a challenge to get everybody on-board, getting everybody to accept full responsibility. Every person in the development process needs to understand the business, how the market evolves, and what competitors are up to. From what they know, they have to create a Minimum Viable product (MVP). This is a central concept in agile and lean philosophies, to not implement too little, not too much, but just about what is takes to keep the customer ahead and the competition behind.



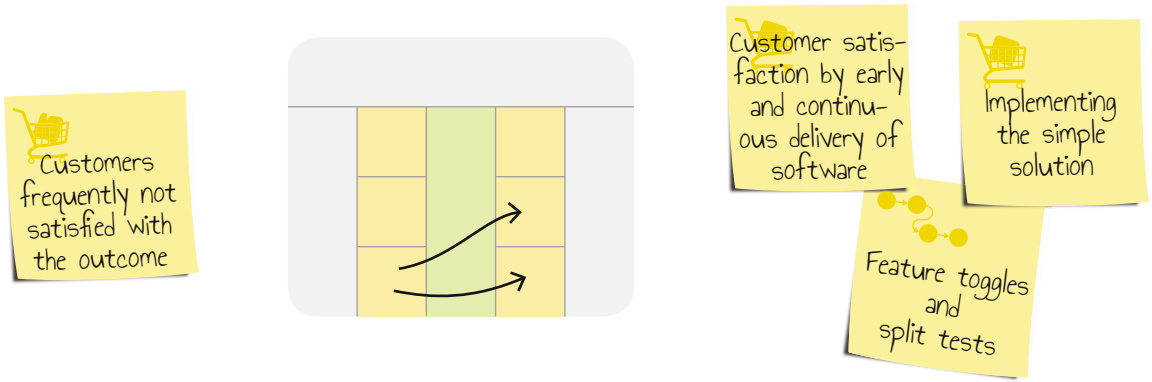
www.dbooks.org

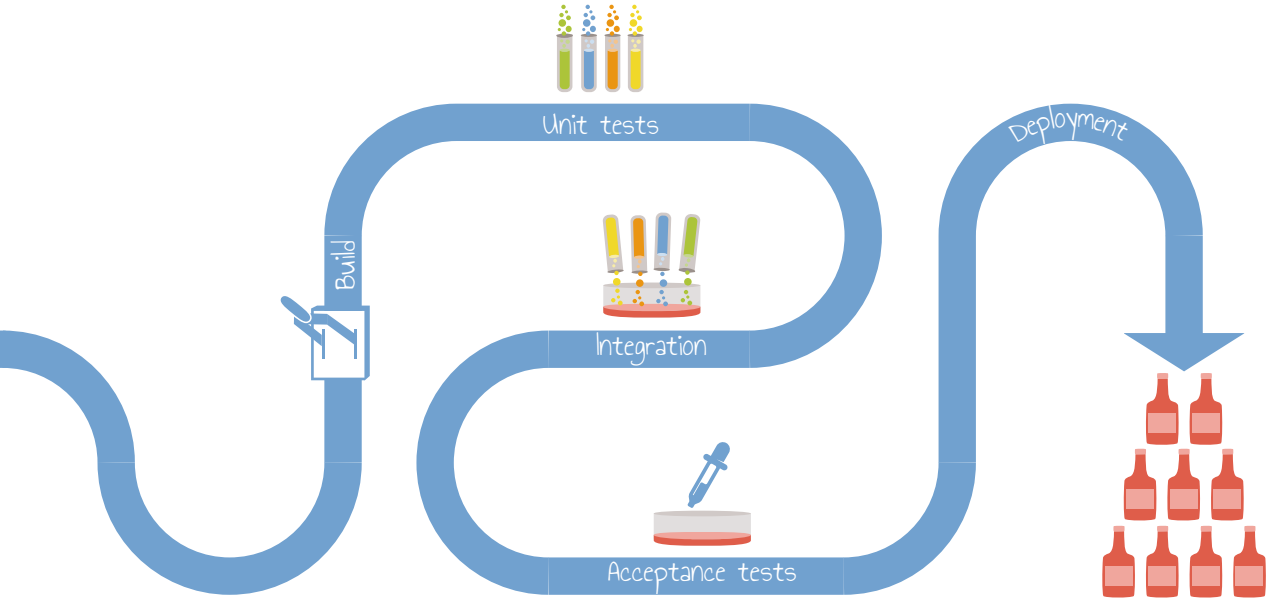
Once a company has established an agile value stream, it becomes easier to differentiate between various products and services. Of course, it's not as simple as just replicating teams, or worse, assigning the same team for several product offerings. Scaling in the product domain, creating new products and services based on existing assets, usually means that the software architecture and supporting systems have to be adapted. Introducing continuous portfolio planning and project visualization will become of utmost importance to remain agile.

Delivering continuously

As innovation and coding can be dealt with by introducing agile methods, so can test, integration, build, and delivery practices. How about eliminating all manual work, developing an ability to deliver a change to the customer within hours, or even minutes? How about being able to deliver bug fixes soon after they are reported, without causing any downtime in development? The solution to this is Continuous Delivery. Its practice is rapidly becoming very popular among major software and service suppliers such as Microsoft, Ebay, Amazon, Facebook, and Google. In particular products that are deployed through the cloud are very amenable to this approach.

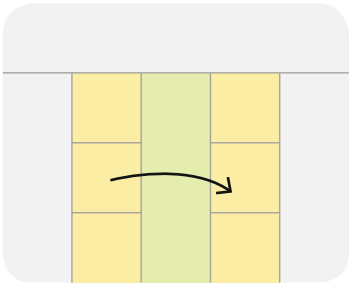
Continuous Delivery helps in becoming more receptive and responsive to customers' needs. To continuously get new features might not be what every customer wish, but many would even love to get early access to semi-finished functionality, to get a chance to give feedback while features are still in development. Feature toggles, to enable unfinished features, are for this purpose a very important concept within Continuous Delivery. Another important concept is to implement essential features in two variants and using A/B testing to determine which variant works best.





Continuous Delivery depends on a delivery pipeline – a series of steps that a product's source code has to go through to be delivered. To establish such a pipeline isn't trivial and doesn't happen over a night. Apart from the very challenging task to automate delivery to customers, the biggest challenge is to automate testing. Constructing a comprehensive test suite that is good enough takes a lot of time, and so does the implementation of the required automation tool support, which is an instrumental and challenging part for any organization. To continuously allow all changes that pass all tests to be deployed to customers requires a mature and stable team that embraces a culture of confidence, humility, and capability. Everything boils down to sound management, to encourage an altruistic mindset and culture, and to build the trust that is needed for such a culture.

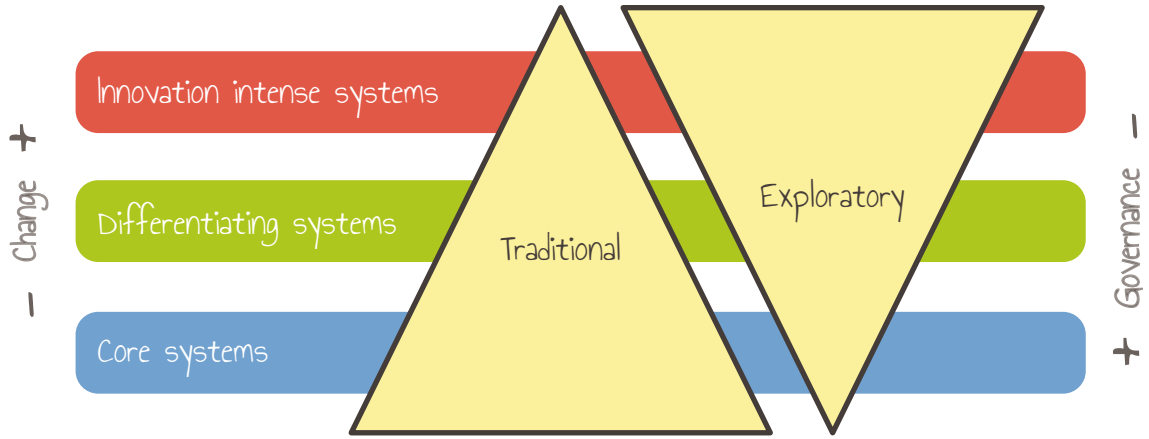




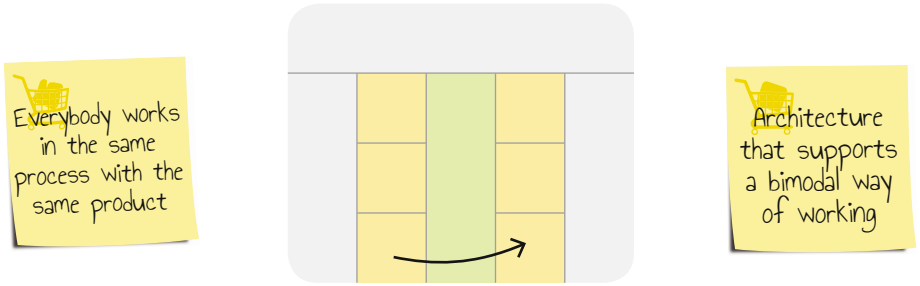


Maintaining stability

When scaling IT systems in an established company, there is usually a need to balance the stability in legacy systems with the flexibility in their agile, bleeding edge offering. The work with core technologies and infrastructure is deliberate, consensus-driven, and slow moving, and so it has to be. The cash cow must not be rushed. The most novel products and services, on the other hand, may be highly experimental and may require more frequent updates in the technology infrastructure.



This is called Bimodal IT, the concept of having two distinct IT methodologies in the same company. The Agile IT team handles the growing needs of the business while the core IT team handles day-to-day business technology functions. The Agile IT team can quickly roll out fast evolving technologies while the core IT team executes long-term plans and goals in a more disciplined fashion.



Keeping the IT systems separated but balanced can be a deliberate and strategic decision. Businesses that have outsourced application development might also want to keep their core IT infrastructure stable in order to keep control of software deliveries from their external partners, who work following agile principles.

Despite the neat separation implied by Bimodal IT, the different teams will need to cooperate. The Agile IT team is still dependent on back office applications provided by the core IT team. Both teams have to respect the different ways of working and agree on mutual processes. However, to avoid getting the teams too intertwined, it's good to create a layered or modularized base system architecture to from the beginning that provides and supports a high degree of change and agility above it.

Keeping two separate IT teams can prove to be a challenge. In order to create a bridge and to be able to prioritize between conflicting interests, it's recommended to have a CIO managing them.

.

www.dbooks.org

When scaling Agile, avoid these common pitfalls:

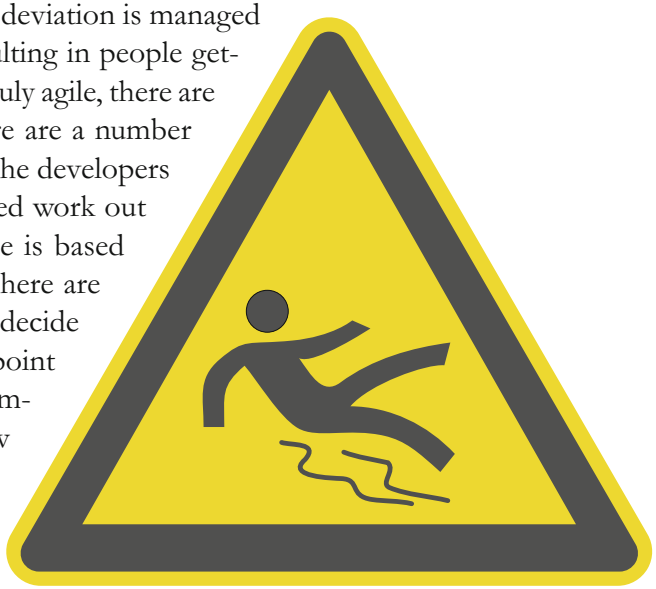
Senior managers often believe they fully understand and embrace agile principles, yet they demonstrate the opposite when making decisions. Situations like this can be difficult to recognize and mend whilst the managers in matter have attended training sessions and use the right agile buzzwords. It's not that they're faking it; they honestly believe they're doing the right things. So what can we do? Experience has shown that having more workshops, training, and discussions are not the most effective approach. A better way to tackle this situation is to let the management team visit another company for in-depth demonstrations and sharing sessions, and to introduce them to agile champions and leaders that can demonstrate what agile leadership really is about.

Low maturity in a company's continuous integration and test procedures causes long quality feedback cycles. To have successfully implemented procedures include at least daily integration and build, and some level of automatic testing. If the environment and processes don't allow for this, then further steps in the agile transformation journey have to focus on this. It's sometimes difficult to identify who should drive this change. There are many stakeholders, including production, legacy system owners, and quality assurance, and this makes running change activities difficult.

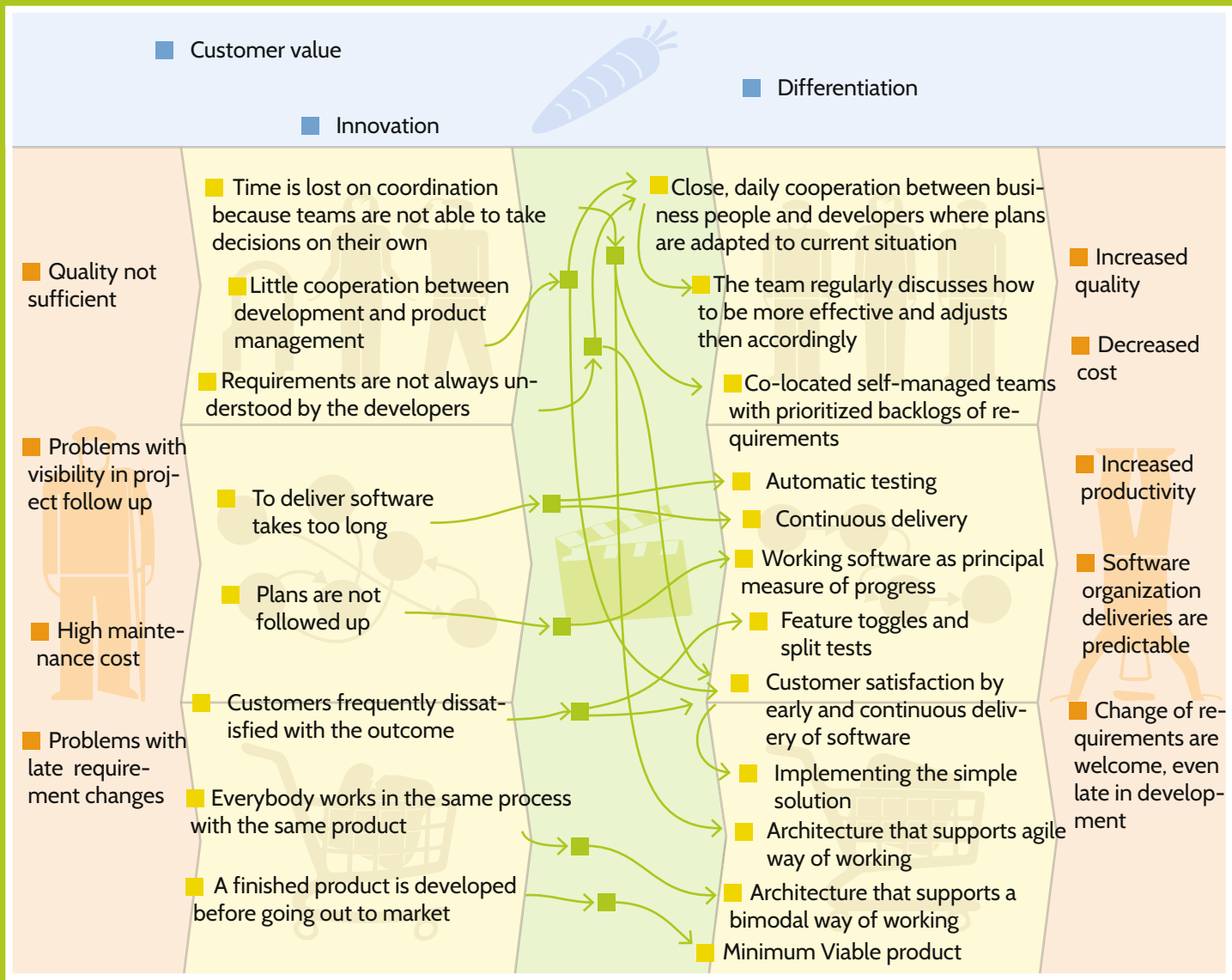
Administrative matters such as organizational structures, governance, forums and roles get more attention than the focus on achieving agility. If development teams aren't agile, then we're scaling something that doesn't work. This causes unnecessary overhead in terms of coordination between teams and micro management of the whole program. So, make sure that development teams work well, that they have product owners, use a backlog and have a clear definition-of-done. Ensure they are self-organizing and have the authority and skills to pull, plan and deliver their work and to get frequent feedback.

Distributed development and dealing with multiple vendors are challenging. Fundamental aspects of agile methods include a team's ability to self-organize, transparent collaboration, and quick feedback cycles. Geographical distance and cultural differences combined with multiple vendors and diverse business and contract models require a lot of the development organization. It's necessary to share visions, high-level goals and agreed-on deliveries between the different organizations, to maintain total visibility. Organizational boundaries have to be removed to get fully cross-functional teams. If you really need to engage multiple vendors, you also need a well-crafted outsourcing strategy that works with multiple suppliers.

Stick to old ways for governing projects with demanding progress reporting hampers agility. With traditional governance – involving steering groups and traditional portfolio management – projects are burdened with detailed estimates in terms of investments and return, milestones and decision gates, and expected dates for deliveries. All results are basically a product of a project with all people and budget allocated in advance. Any deviation is managed by renegotiating project scope and budget, resulting in people getting shuffled between projects. When working truly agile, there are no projects with allocated people. Instead, there are a number of backlogs and a total development capacity. The developers organize themselves in teams that pull prioritized work out of these backlogs. All planning and governance is based on a quarterly, monthly, and weekly cadence. There are still people responsible for the business. They decide what and when to release. From a management point of view, this regime requires a lot of trust. It's important that there is full transparency to how much the teams can deliver per delivery. This enables product owners to make forecasts and reprioritize the next delivery.



www.dbooks.org


Customer value
Innovation
Differentiation
Quality not sufficient
Problems with visibility in project follow up
High maintenance cost
Problems with late requirement changes
Time is lost on coordination because teams are not able to take decisions on their own
Little cooperation between development and product management
Requirements are not always understood by the developers
To deliver software takes too long
Plans are not followed up
Customers frequently dissatisfied with the outcome
Everybody works in the same process with the same product
A finished product is developed before going out to market
Close, daily cooperation between business people and developers where plans are adapted to current situation
The team regularly discusses how to be more effective and adjusts then accordingly
Co-located self-managed teams with prioritized backlogs of requirements
Automatic testing
Continuous delivery
Working software as principal measure of progress
Feature toggles and split tests
Customer satisfaction by early and continuous delivery of software
Implementing the simple solution
Architecture that supports agile way of working
Architecture that supports a bimodal way of working
Minimum Viable product
Increased quality
Decreased cost
Increased productivity
Software organization deliveries are predictable
Change of requirements are welcome, even late in development

Get inspired

This scenario has been based on case studies of different companies that have made this journey, to scale with Agile. Learn from their experiences, what they gained and what they had to overcome.



Pruning a bush

At this company, any development of new functionality literally drowned in the work to support customers and fix bugs. Read about how they implemented SCRUM and Kanban without jeopardizing the relations with their customers and quality in their old products.



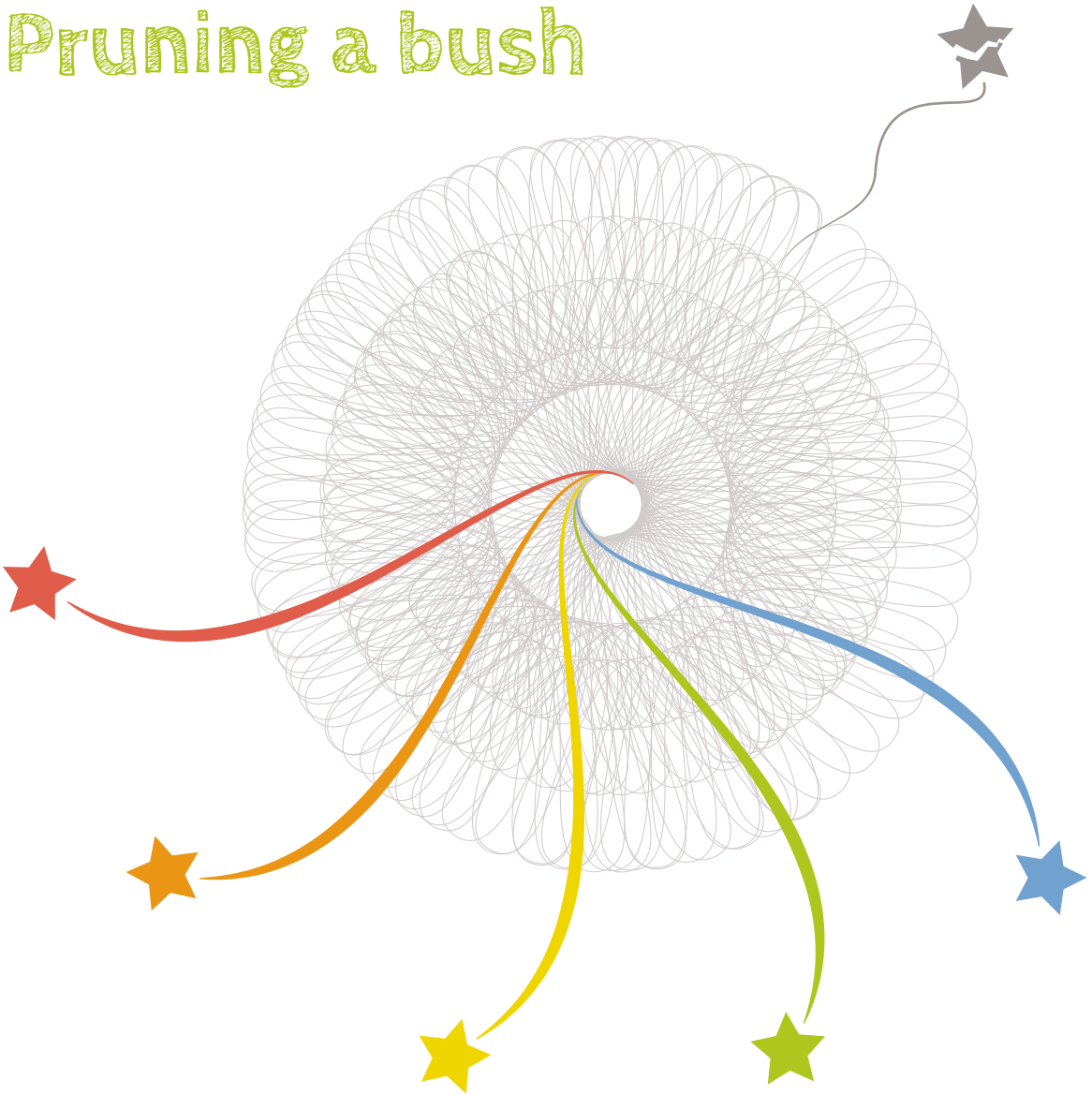
Ensuring prima deliveries

Critical bugs passed unnoticed for weeks or even months. Corrected bugs took days or weeks to deliver, since the procedure is manual and requires the system to be shut down This was the situation when a team at Prima decided to implement Continuous Delivery.

www.dbooks.org

CASE STUDY / Deliver 24/7

Pruning a bush



The company had long release cycles due to their lack of capacity but mostly because their customers do not want changes too often. Many products in the field never even got updated, products that the company's technicians needed to be able to maintain and service. A few of the larger customers desired to have the products branded as theirs. The consequence was a complex branching strategy (a way to keep track of the software for each customer). Every bug needed to be fixed and tested in all of those branches. This made the releases very difficult. The many branches made it impossible to keep their release deadlines.

Any development of new functionality literally drowned in the work to support customers and to fix bugs. The development department was drenched in work and very little new software was produced. However, a new product was in the roadmap. The organization knew they needed to make this a priority without jeopardizing the relations with their customers and quality in their old products.

The many and conflicting patches made integration and verification very difficult. The team had to move all patches from the release branch to the main branch. Sometimes there were several release branches to merge into the main branch. It was a mess.

The maintenance team added the patches directly on the customer's production branch at any time. So when a new release was created, all patches from the different customer branches had to be added to this release. The many and often conflicting patches made integration and verification very difficult. Having released, the team then had to move all patches from the release branch to the main branch. Sometimes there were even several release branches to merge into the main branch. It was a mess.

The Agile transformation started by training the development teams, product owners and managers in Agile, Scrum and Kanban methodologies*. After the training was completed, new ways of working were introduced in a big bang. The department was divided into three development teams. The team that got responsibility for maintenance started to work according to Kanban. The team that got to work with customer projects and the team that got to work with new prod-

* www.scrumalliance.org
kanbanblog.com/explained

www.dbooks.org

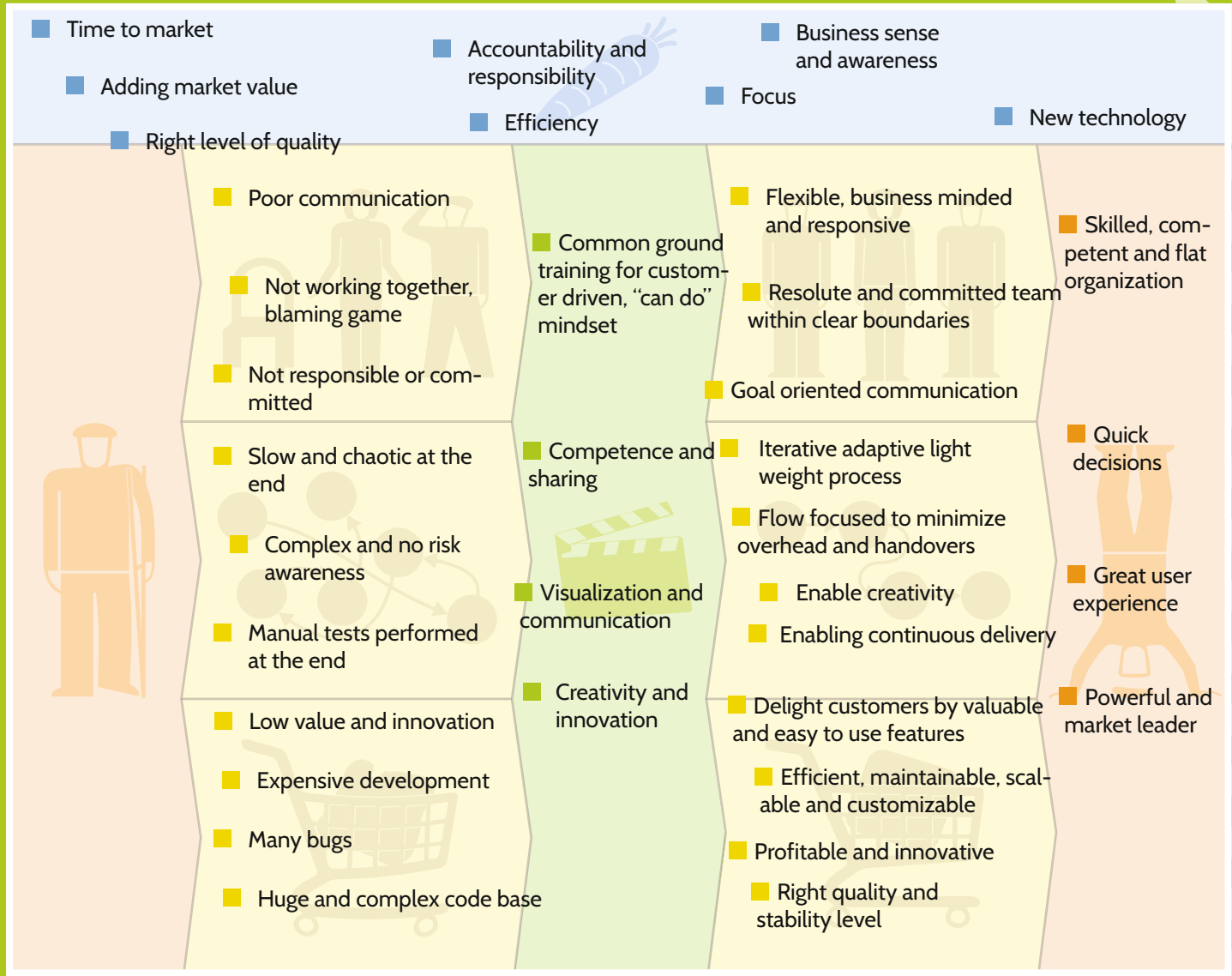
ucts started to work according to Scrum. The people in the test team were all distributed in the agile teams. They were assigned to write user stories, define acceptance criteria and also to train the developers in how to test. The testers could now focus on more complex test cases and the overall quality.

External coaches facilitated the first sprint. The responsibility was then handed over to the team's Scrum masters and Kanban leaders. The two development teams adapted quickly and started to produce and solve challenges as they arose. The external coaches were still available, to support the teams, the scrum master and the managers in their new roles. They also got help with their group development, to help them in seeing how to improve. The branching strategy was changed after a thorough analysis. Now, all bug fixes and new functionality had to be tested and merged into the main branch after every sprint.

As the maintenance team started to work with smaller batches of bug fixes, they could as well take the step and merge corrections both into the service pack release branch and the main branch. Integration and verification of every batch went so much faster. They could in this way guarantee the quality on both branches.

It turned out to be very important to not exceed 10 bug fixes in a batch. Twice, the tried to include 20 bug fixes. But after having discussed the effects in retrospective, they concluded they ought to stick to 10. With 20 bug fixes, it simply took too long time to do the integration and verification. By sticking to 10, the releases could be finished on deadline and without any over-time. They hadn't been able to do this for a very long time, a huge step forward according both to management and the employees.

To get the software departments up and running with Scrum and Kanban required two weeks of initial training.



www.dbooks.org

CASE STUDY / Deliver 24/7

Ensuring prima deliveries



Many organizations are suffering from sub-optimal development and deployment processes. Common bottlenecks are: outdated tools, lack of systematic and pro-active error handling procedures, and deployments that only can be made at night time in order to limit downtime. The consequences are low quality and delayed deliveries. The feedback loop from customers to developers tends to be too long. Critical defects can pass unnoticed for weeks or even months. With manual deployment procedures that require a system to go offline, defect fixes might take days or weeks to deploy.



This was also the situation when the Swedish company *Projekstyrning Prima* (Prima) decided to implement Continuous Delivery of its route planning systems. To that end, Prima developers introduced new systems for source control management, build automation, automated deliveries, and a new way to log information – one module and one developer at a time. These steps made a great difference to their productivity, their lead times to correct bugs and their confidence in their products. It was no longer necessary to take systems offline for an update, or to work long nights or weekends. With this set of tactical changes, new product versions could be deployed in the middle of the day, without any downtime. The resulting shorter cycles and faster deliveries of both features and defect fixes led to increased customer satisfaction and fewer calls to Prima's support lines. The shorter cycles also made it easier to plan the work ahead on a realistic time scale, and measure effects such as product quality of their new approach.





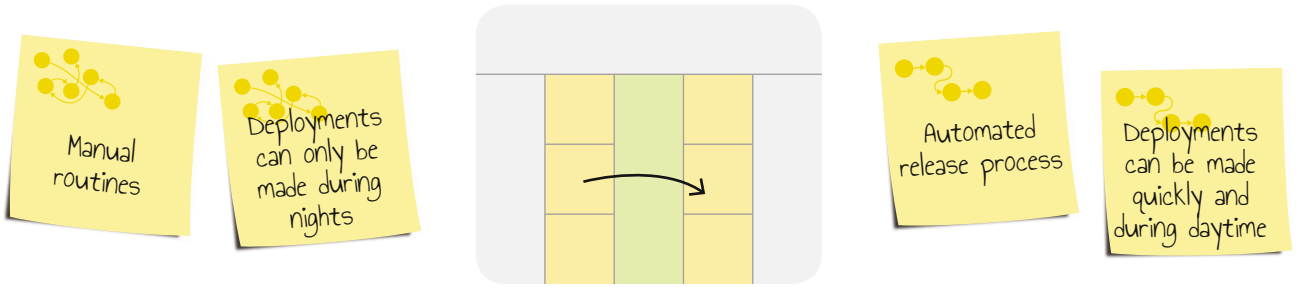
The key driver for Prima was to increase the release cadence without jeopardizing the quality and stability of the product. To make this possible, the development pipeline had to be fully automated, which itself required some significant updates of their existing tool chain.

www.dbooks.org

Cloud platform Microsoft Azure

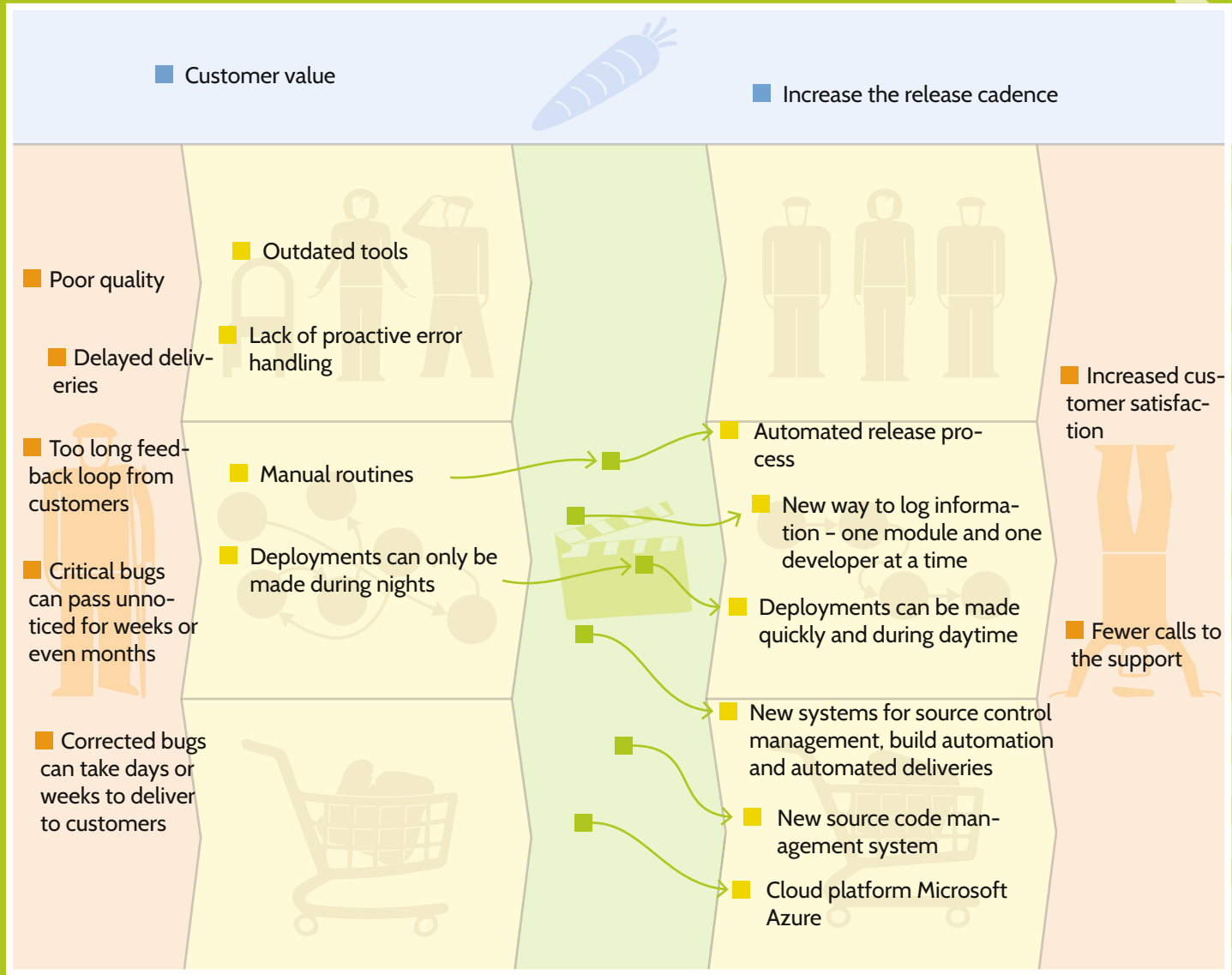
Prima was using the Microsoft Azure cloud platform, but this wasn't a key problem. The challenges they ran into could have arisen with any cloud technology stack. While cloud technology certainly facilitates a transformation to continuous delivery, this wasn't a key requirement.

The changes that the Prima team made took just over three weeks – less than 16 days to be precise. Some of the major improvements were made by rethinking and simplifying the product release procedures. Another cornerstone for making improvements was to introduce monitoring systems: detecting when things go wrong is essential to continuously improving processes and removing bottlenecks. This continuous improvement is key in true enterprise agility and can also be traced back to the Lean Thinking philosophy. Continuous improvement is not the destination—it is the journey itself.



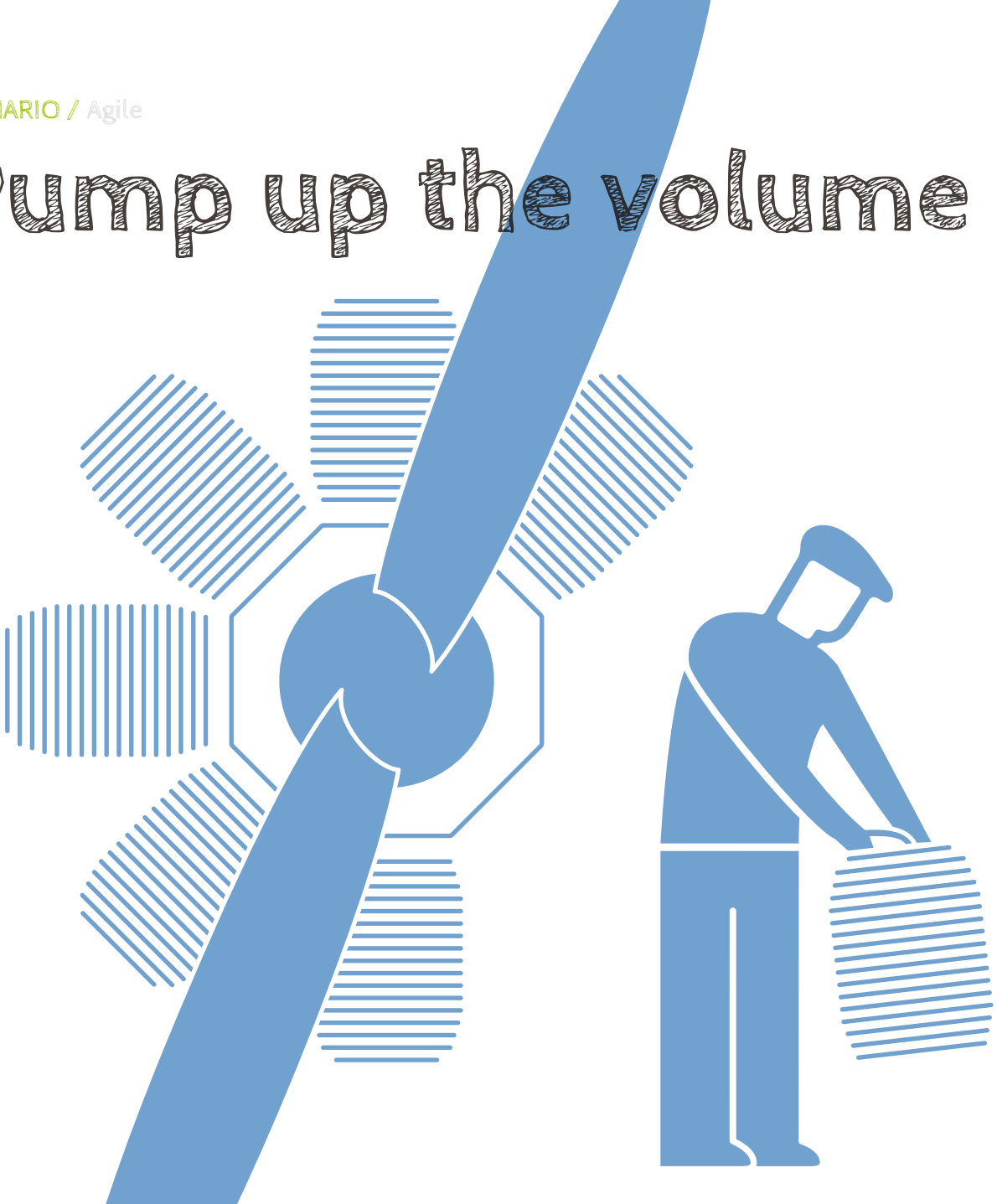
New source code management system

Lean Thinking is derived from the Toyota production System, which itself contains many concepts using Japanese words. One simple tactic is described by "Genchi Genbutsu", which refers to the simple act of going to the production floor to see "what's going on." By simply letting a coach walk through the steps with every member of the team helps to overcome many problems. Soon it became clear that the chance to succeed increases if everybody knows the whole team, and the product, before embarking on this journey. Of course, using support from modern tool chains is a necessity. These tools don't need to be very expensive—many of the tool chains are available free of charge as open source products. While some tool migrations might take some effort, such as migrating to a modern source code management system, such investments will pay off in the end. This is one of the many trade-offs that teams will have to make in a process improvement initiative.



www.dbooks.org

Pump up the volume

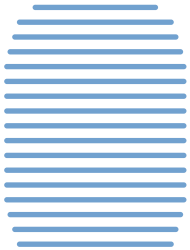


> The scaling Agile model is for companies that want to start working with Agile on a larger scale. The complete model describes scaling in three domains: size, offerings and value stream.

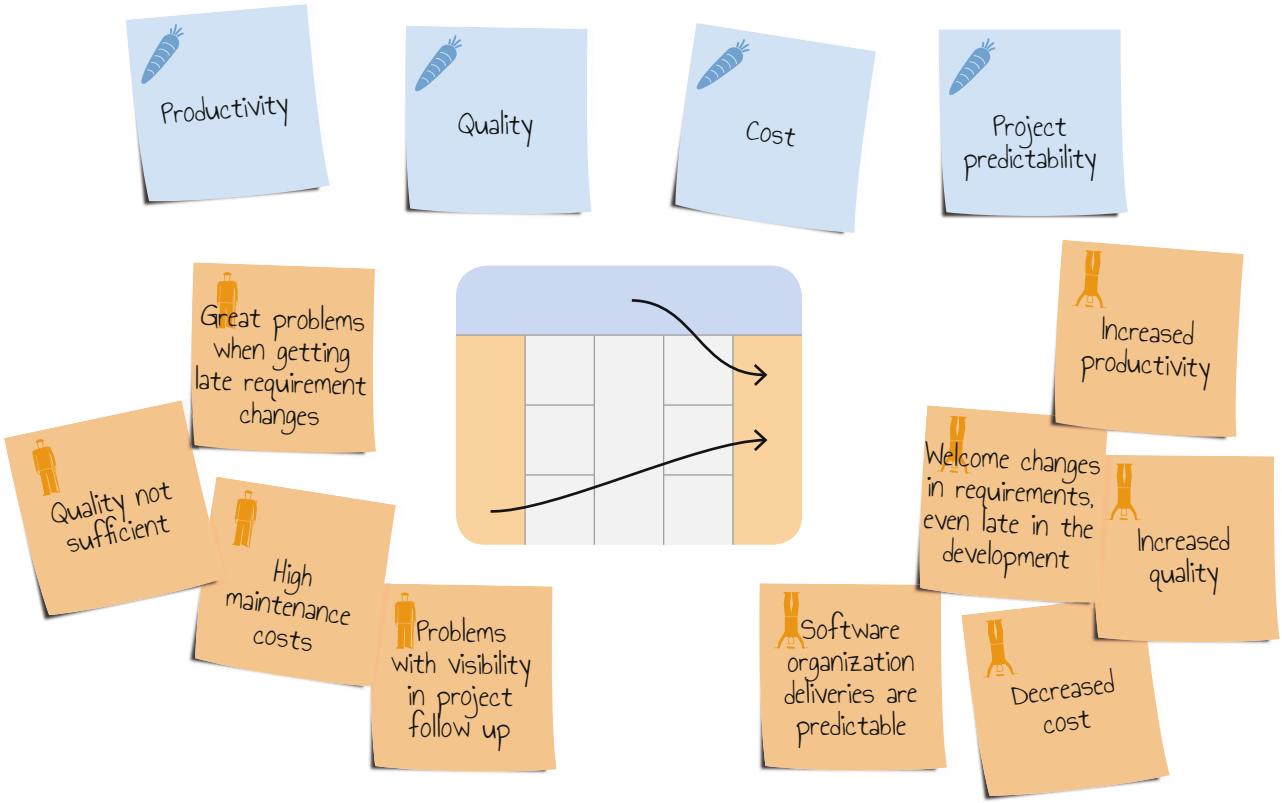
This chapter focuses solely on the size domain.

The model is for companies that aim to extend the number of people in the value stream so that more teams work together towards a joint delivery. The typical starting point is that a development department has been using Lean and Agile successfully for a few years, and now they wish to spread the ways of working through the rest of the company.





www.dbooks.org



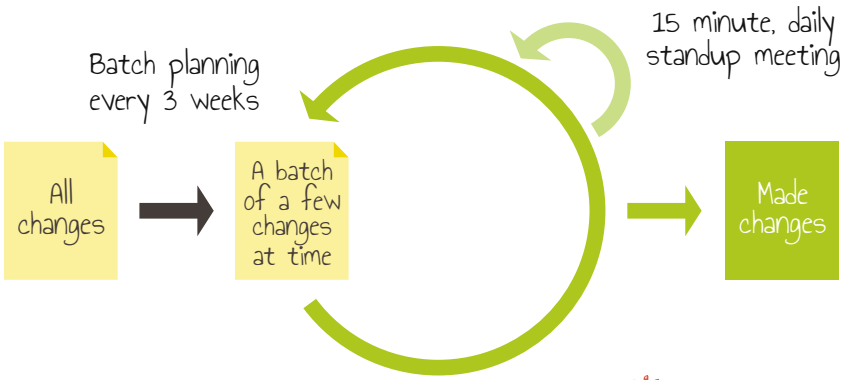
When scaling a business in terms of efficiency, all that matters is how to build the products faster, more predictable and in bigger volumes. Innovation is not a matter at this point. This is a business need that mostly appears in very large companies that deliver complex products or services to the market. They search for ways to be more productive together in an organization that suffers from too many dependencies between products, services and departments. This

complexity makes the offering sensitive to changes and the organization likely spends significant resources on maintenance. What complicates everything is that the scaling would need to be made without serious interrupts in the product development. This is where scaling Agile comes in, an approach cut out for the change.

Agile development, for example Scrum and Kanban*, has rapidly proven to be the preferred methodologies among software teams and their developers. Even if a company formally hasn't made the switch to use Agile methods, it's not unlikely that some of their teams already have started to work this way. Teams working in an Agile fashion strive for clear goals and boundaries, open communication with full visibility of decisions and priorities.

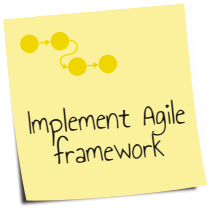
Letting go of details and focusing on Lean and Agile principles will turn out to be challenging for most managers in a traditional organization, even if it's been successfully proven empirically. They have to be courageous and trust in the method and in the staff they manage. In fact, Agile provides discipline, transparency and working code frequently so trust will come by itself. It's important to grasp the idea that scaling Agile has no end; this is the way the organization is run.

Organizations that have been scaled in this way shows that they are able to deliver utterly complex products and services with remarkable efficiency. There are of course pitfalls ahead, when focusing on efficiency over a too long period of time. While being reactive is considered good in Agile, it's important to consider all customer requests carefully. The product owner has to make the right priorities. It's important to not forget to start innovate again.

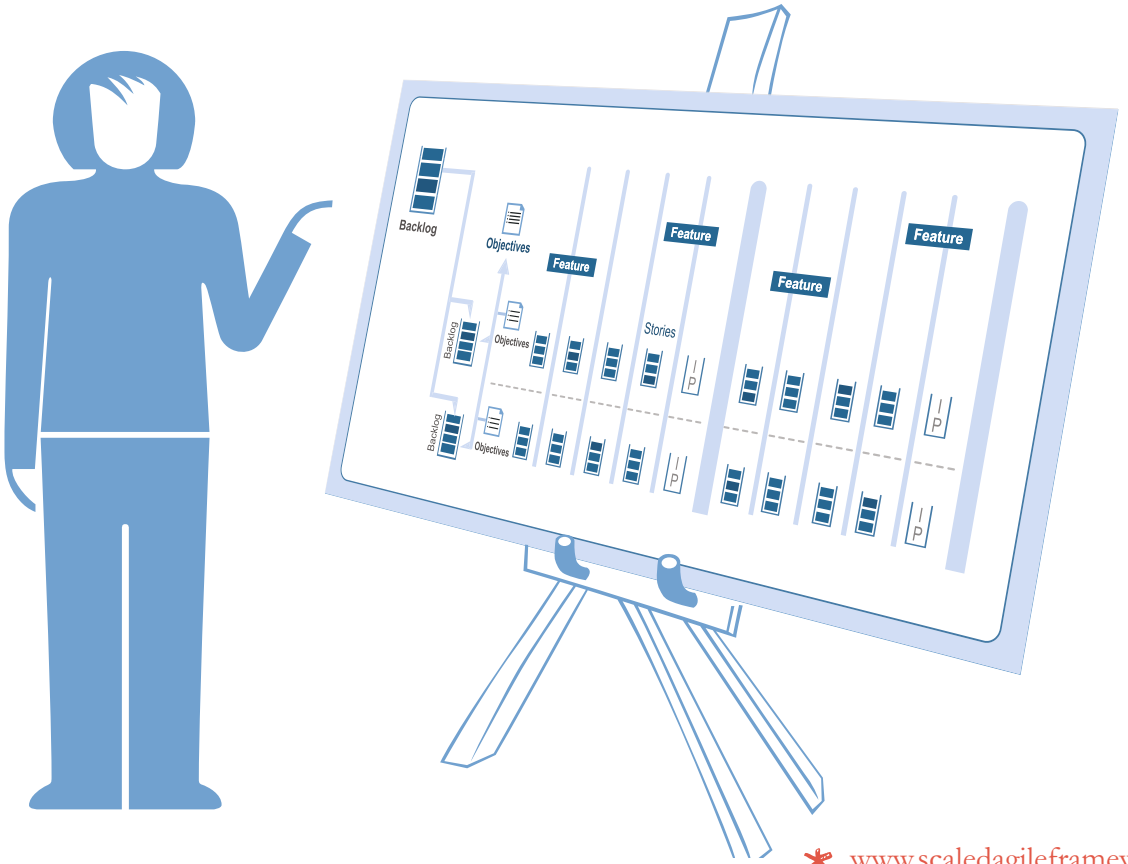


* www.scrumalliance.org
kanbanblog.com/explained

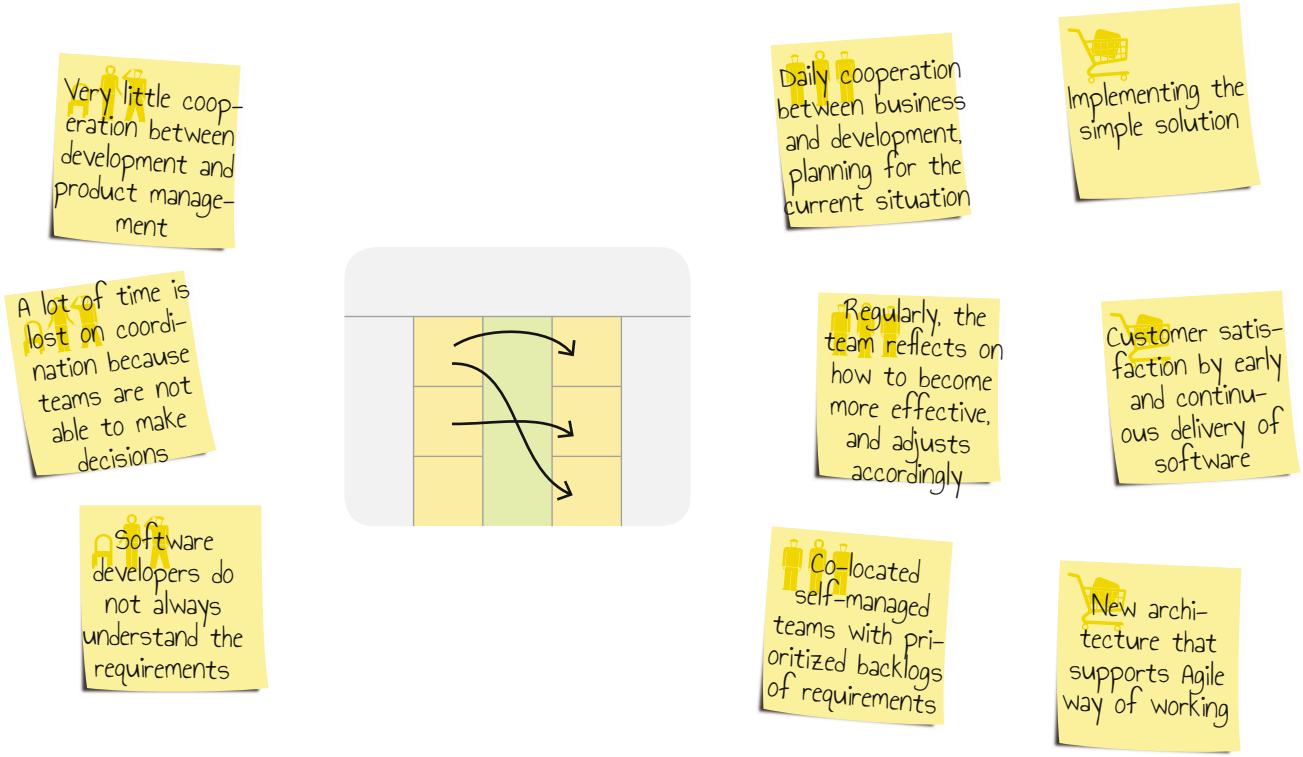
www.dbooks.org



There are several frameworks for scaling Agile and there is no right or wrong choice. A decision of what combination of frameworks to use can be made once there is an agreement of what is relevant to the organization. If you already use Scrum in the company, you might want to consider LeSS (Large-Scale Scrum). LeSS is sprung out of complex R&D development and emphasizes on continuous learning, inspection and adaption of both product and processes from a systemic point of view. If portfolio and program level management is central to your company, you might want to get a closer look at SAFe (Scaled Agile Framework*). SAFe put emphasis on governance, program and portfolio management and in particular suitable for organizations from hundreds to thousands of developers.



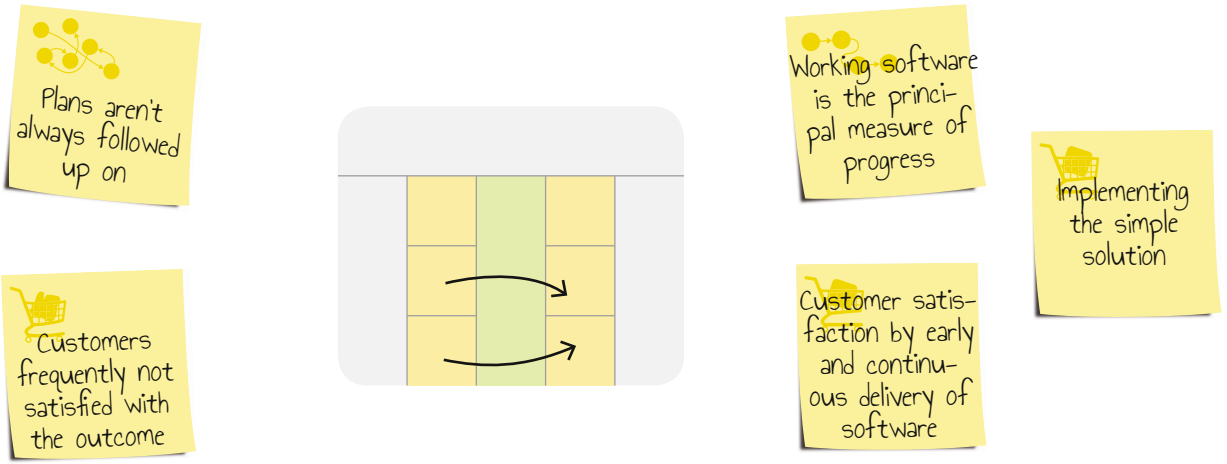
* www.scaledagileframework.com



All frameworks have their differences but they all share the Agile principles. This means also that the change will require new roles and ways-of-working, even in an organization that uses Agile development methods. It's a good idea to form a cross-functional change team. To change the mindset of leadership and the governance will be one of the biggest challenges in this transformation. One of the most important principles of Agile software development is to have

www.dbooks.org

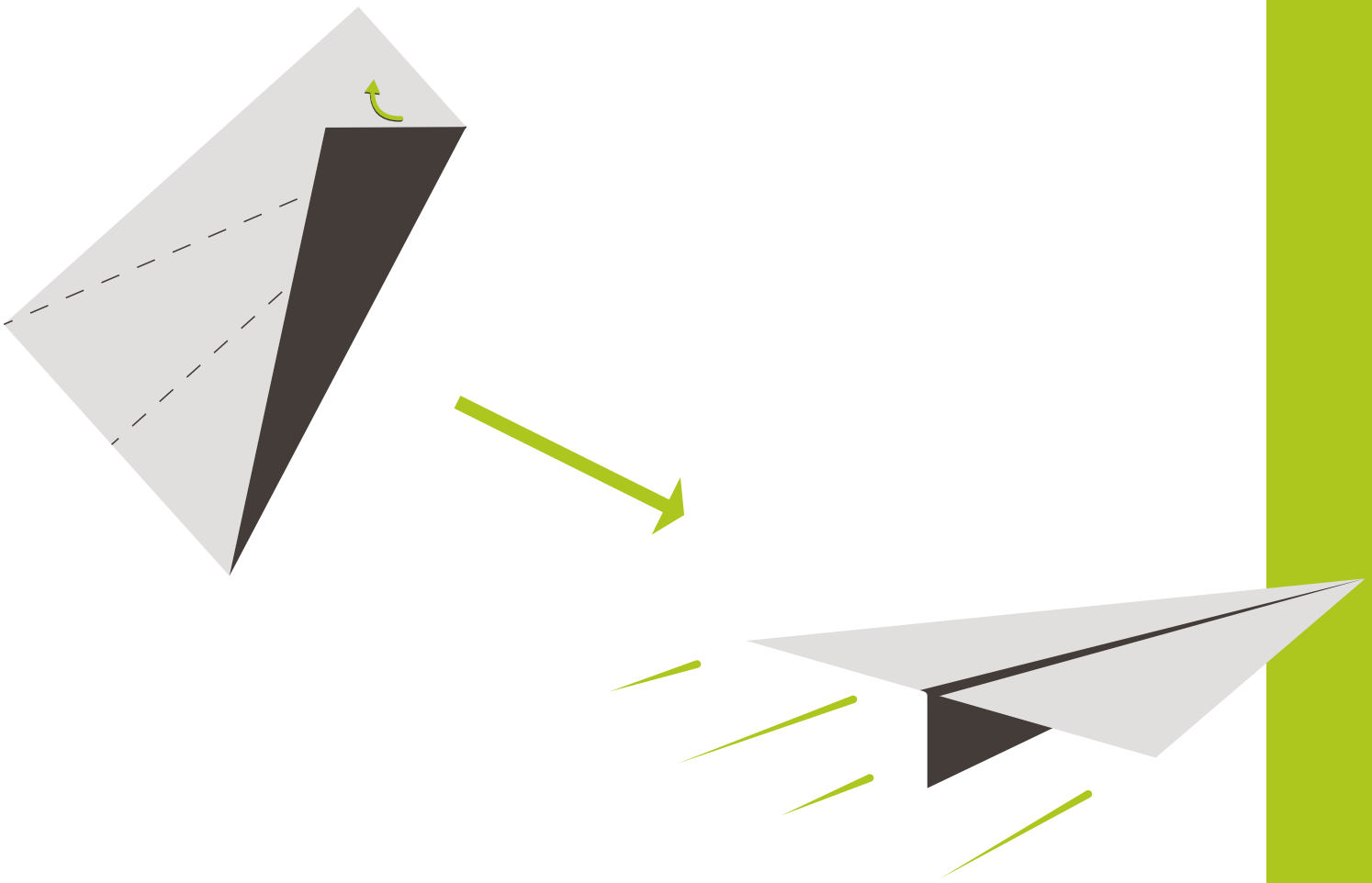
self-managed co-located cross-functional teams working. The teams should understand the requirements fully and have all necessary competence to take all decisions for the functionality that they are responsible for. Read more about change in **Your journey**.

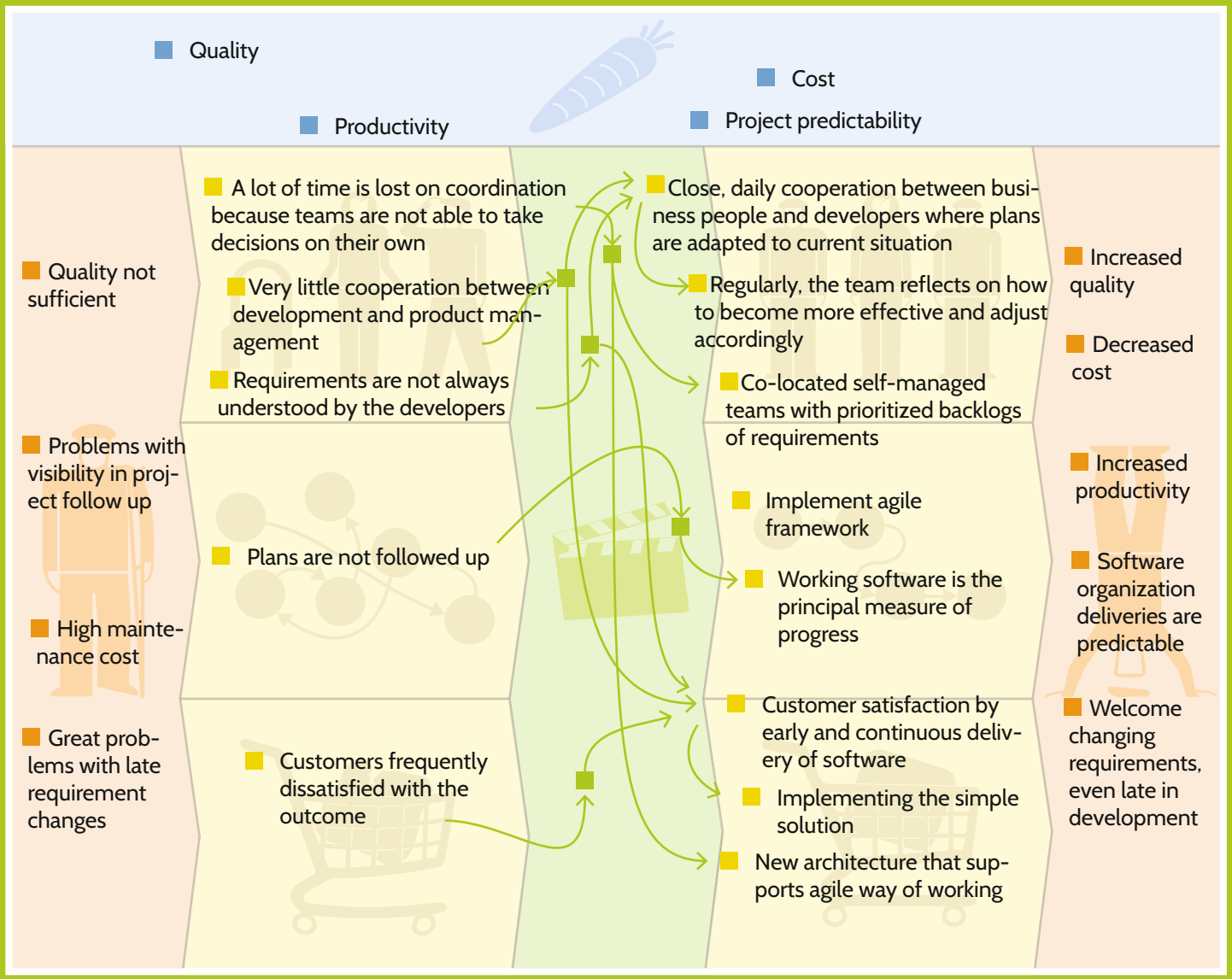


Another important aspect is to have close cooperation between the product management and the development team. Ideally product management competence must be in the self-managed teams, so that the communication is fast and direct. But there are some Agile frameworks that promote to have the product management in a central team. Whichever way this is organized communication needs to be daily. Teams need to regularly reflect on how to become more efficient. This should ideally be done regularly.

Follow up on progress in software-based projects is usually a big challenge. In an Agile project the progress is measured by code that is actually working (according to the definition of Done). Since functionality is split up in smaller pieces (chunks), this is a proven way to measure progress. This also drives customer satisfaction, due to that customers can give their feedback in early

phases of the development cycle instead of at the end of a project. Customers usually want specific features while engineers like complicated implementations that cover everything. This is why emphasis is put on choosing the simplest possible solution. It makes customers happy and engaged already in the early phases of the software development lifecycle.




Quality
Productivity
Cost
Project predictability
A lot of time is lost on coordination because teams are not able to take decisions on their own
Very little cooperation between development and product management
Requirements are not always understood by the developers
Close, daily cooperation between business people and developers where plans are adapted to current situation
Regularly, the team reflects on how to become more effective and adjust accordingly
Co-located self-managed teams with prioritized backlogs of requirements
Implement agile framework
Working software is the principal measure of progress
Customer satisfaction by early and continuous delivery of software
Implementing the simple solution
New architecture that supports agile way of working
Plans are not followed up
Customers frequently dissatisfied with the outcome
Quality not sufficient
Problems with visibility in project follow up
High maintenance cost
Great problems with late requirement changes
Increased quality
Decreased cost
Increased productivity
Software organization deliveries are predictable
Welcome changing requirements, even late in development

Get inspired

This scenario has been based on case studies of different companies that have made this journey, to scale with Agile. Learn from their experiences, what they gained and what they had to overcome.



Global R&D goes agile with SAFe

This study from Ericsson brings up the difficulties in implementing GoAgile with SAFe in a global organization.

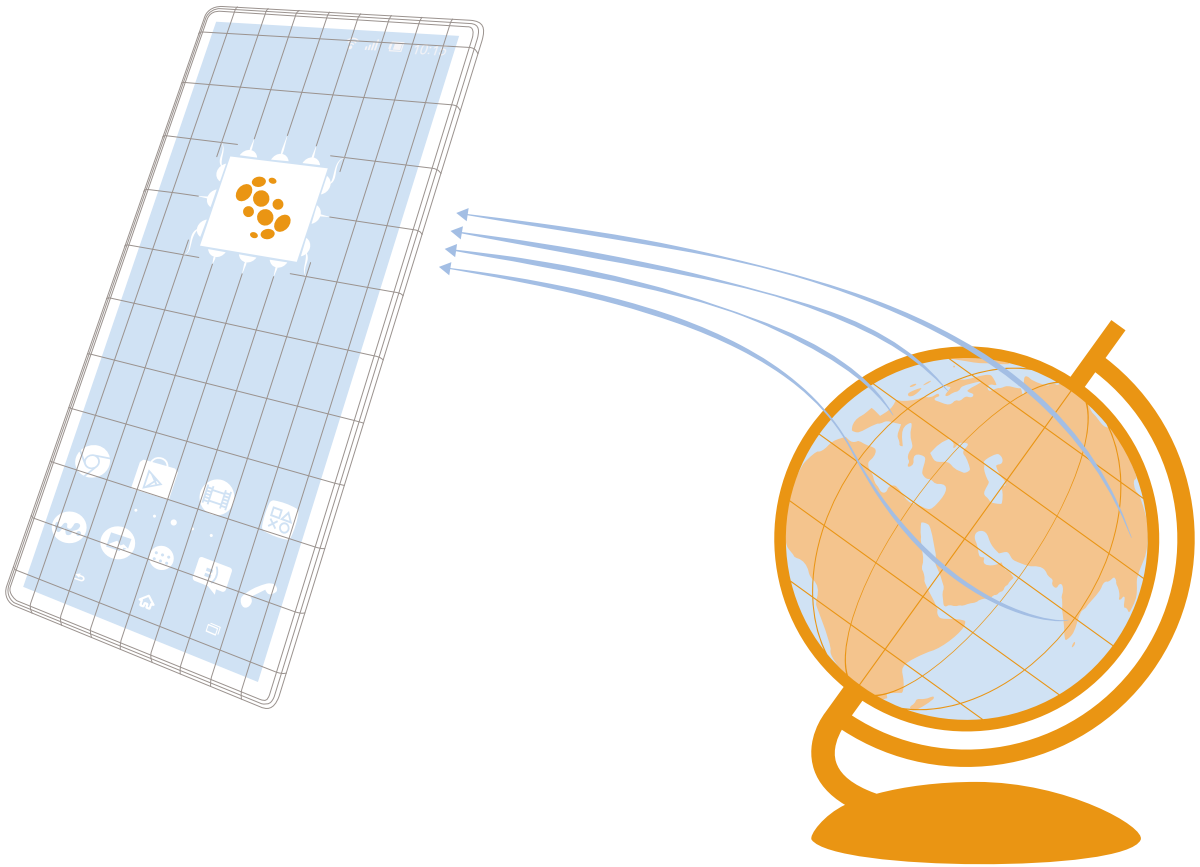


Multi-site development

If you're into services, you might want to read about how the large mobile operator gained excellent predictability by visualization.

www.dbooks.org

Global R&D goes agile with SAFe*



* www.scaledagileframework.com

The global high tech company had struggled for quite some time to deliver as promised. This had created a strained relationship between the company and their customers, which seldom believed in the promised dates and the quality of the deliveries. When the customers got back to the company they had to wait far too long to get corrections. The market competition was at the time getting tougher and tougher for the company. Many competitors aspired to be the rising star. The company had to realize that they have to be much more efficient. They had to make the most out of their employees to speed up development and to solve customer feedback cases.

These drivers initiated their Agile transformation:



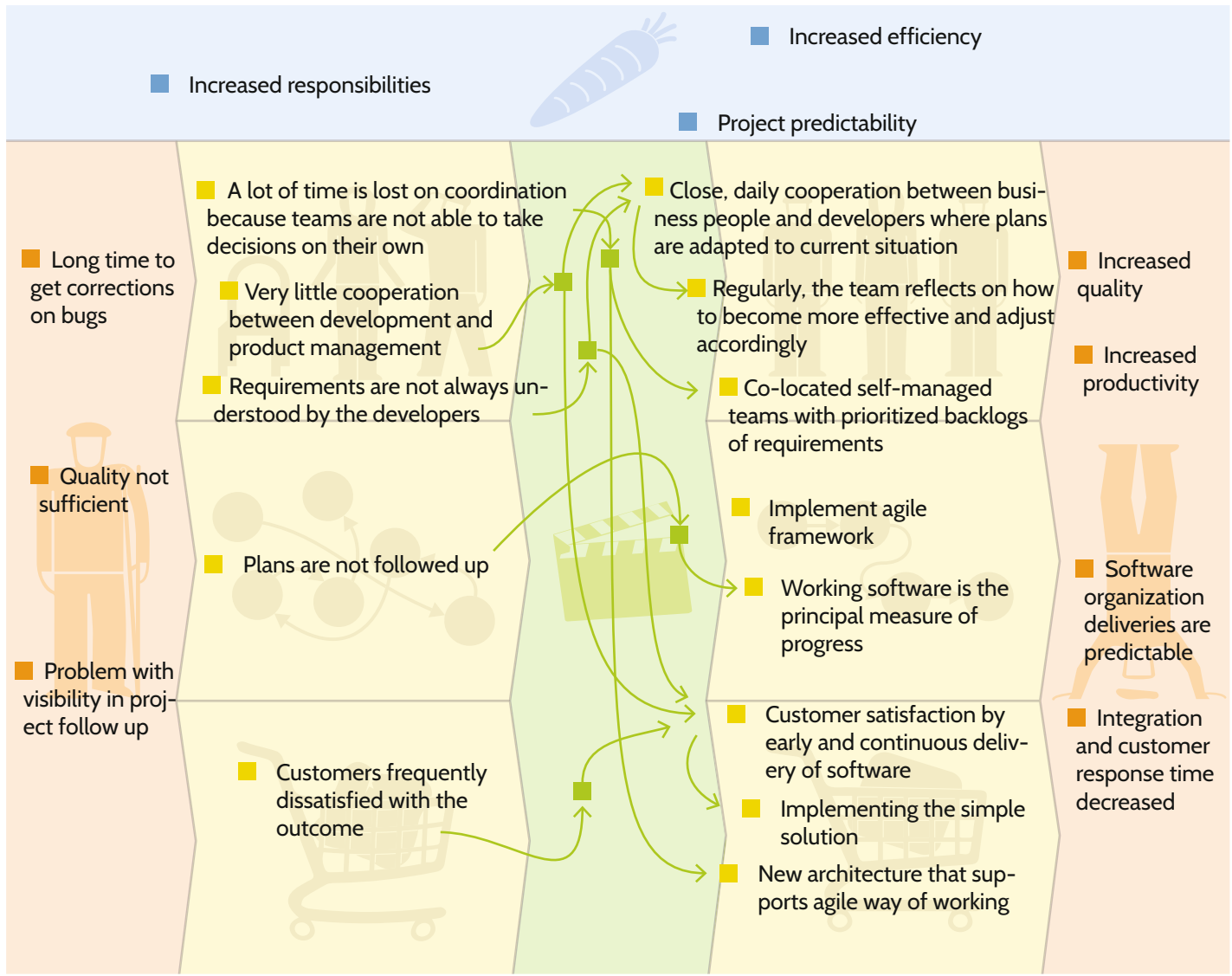
The first ideas of how to structure the organization were formulated in 2012. The focus was on speed and how to fight competition from other platform providers. The transformation journey started in mid 2013, by the creation of an organization that would fulfill the demands from the current mobile platform market.

The organization was set up as an R&D organization parted in four requirement areas (RA) and one integration & verification organization. Each RA was responsible for either a technology area (like core SW) or a function area (like test). These RAs were located to 4 different sites over the world; in fact, the same RA could even be distributed over several different sites. About 800 people in Sweden and about 1500 persons globally were affected by the transformation. The change was led by a core team working in an Agile way, using whiteboards and visualization.

www.dbooks.org

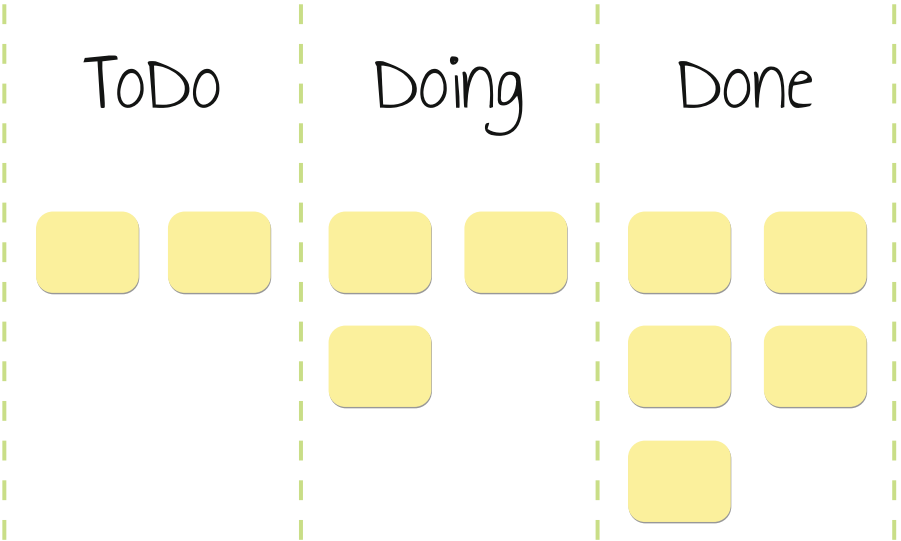
The R&D organization improved well, both the integration time and the customer response time decreased. A major reason this went so well was that the introduction of Continuous Integration and Continuous Delivery. Read about this in the chapter **Deliver 24/7**. Another reason was the coaches, who worked with the teams to help improve transparency and collaboration. A stable change velocity helped the product management to make releases predictable. Realizing that more and more Agile tools actually worked, they completely changed their mindset. A year after the start of the transformation, the company was able to manage a full program increment, an activity that last over a quarter of a year. At this point in time, everybody in the organization could go to the visualization room and have a look at the different RA plans, goals and KPIs. Everything was updated on a regular basis.

The biggest challenge in the transformation was to change the mindset in the organization. Slowly, small success stories spread in the company, making people to start trust one another. A cultural change like this takes a long time and needs to be given sufficient focus.



www.dbooks.org

Multi-site development



Productivity

The management of a large mobile operator had realized that one of their biggest projects was not progressing at all. The project had been planned as always, in accordance with waterfall principles. All specifications had been created and yet no one knew how to start. With lots of money and prestige already invested in the project, it simply had to succeed.

The project aimed to merge a multitude of different systems, of different age and status, into one big system. Part of the development was made in-house, but part were also made by many vendors spread over the world.

In their current system, it was difficult to find knowledge about their customers. The information was spread out over different systems and customer service needed to access all these systems to support their customers. But, the work took a long time and it was hard to train them. The system was practically impossible to work with. It was particularly difficult to create bundled products across different channels. The tools needed existed, but resided in systems that were not connected. They desired to create a cohesive experience across all channels. It should look and feel the same on all platforms.

Quality

They finally understood that such a massive and complex project could not be thoroughly planned from the start to the end. A clear goal had to be set and the solution had to develop over time, with tight learning and feedback cycles. That's when they decided to start working with an Agile methodology.

Project predictability

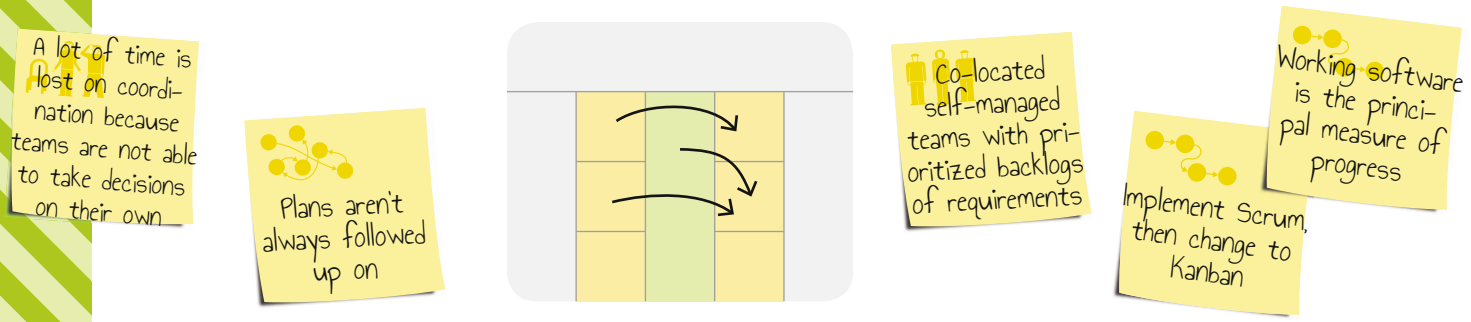
First they implemented Scrum as a project methodology. They divided the large in-house team into two smaller teams, to keep the team members focused. A single, prioritized backlog was created and the teams started to develop based on it. This change alone turned out to be one of the most important ones they ever made in the project. During the following six months, an extensive recruiting took place. In the end, five teams worked side by side on the same project.

But one challenge remained. They didn't manage to solve the long lead times that were required for each new function they added. The many dependencies between teams and vendors made it very difficult and many functions had to be iterated a few times before they worked. Velocity, by which each team was measured, turned out to be useless as a mean to estimate the releases. Dependencies were handled during the refinement of a task, by pre-order from the team that depended on certain functionality. When a task was developed based on the specifications, it often had to be redone. The team that ordered the functionality in the first place would then often find something that needed to be changed, and place a new pre-order.

To have any chance to succeed, they needed to reduce the lead time and the significant amount of ongoing work. The change team concluded in that Scrum had to be replaced by Kanban, in each team and on a project level. Work-in-progress limits is the key principle in Kanban. Additionally,

www.dbooks.org

a limit was set of how much ongoing work was allowed at the same time. On project level, a Kanban board was introduced. It gave an overview of the features each team was working on. A clear "definition of done" was also defined between each team. The project management could now use this board and start estimate throughput and lead times on an overall level. They had real data to analyze in order to see if the project was going to manage the project deadline. They could now



re-prioritize based on this information. Each team and vendor got such a Kanban board, enabling them to prioritize tasks and solve their bottlenecks. They also introduced collaborative analysis and design, in which key people from each team met and talk through each task, what the task means to them.

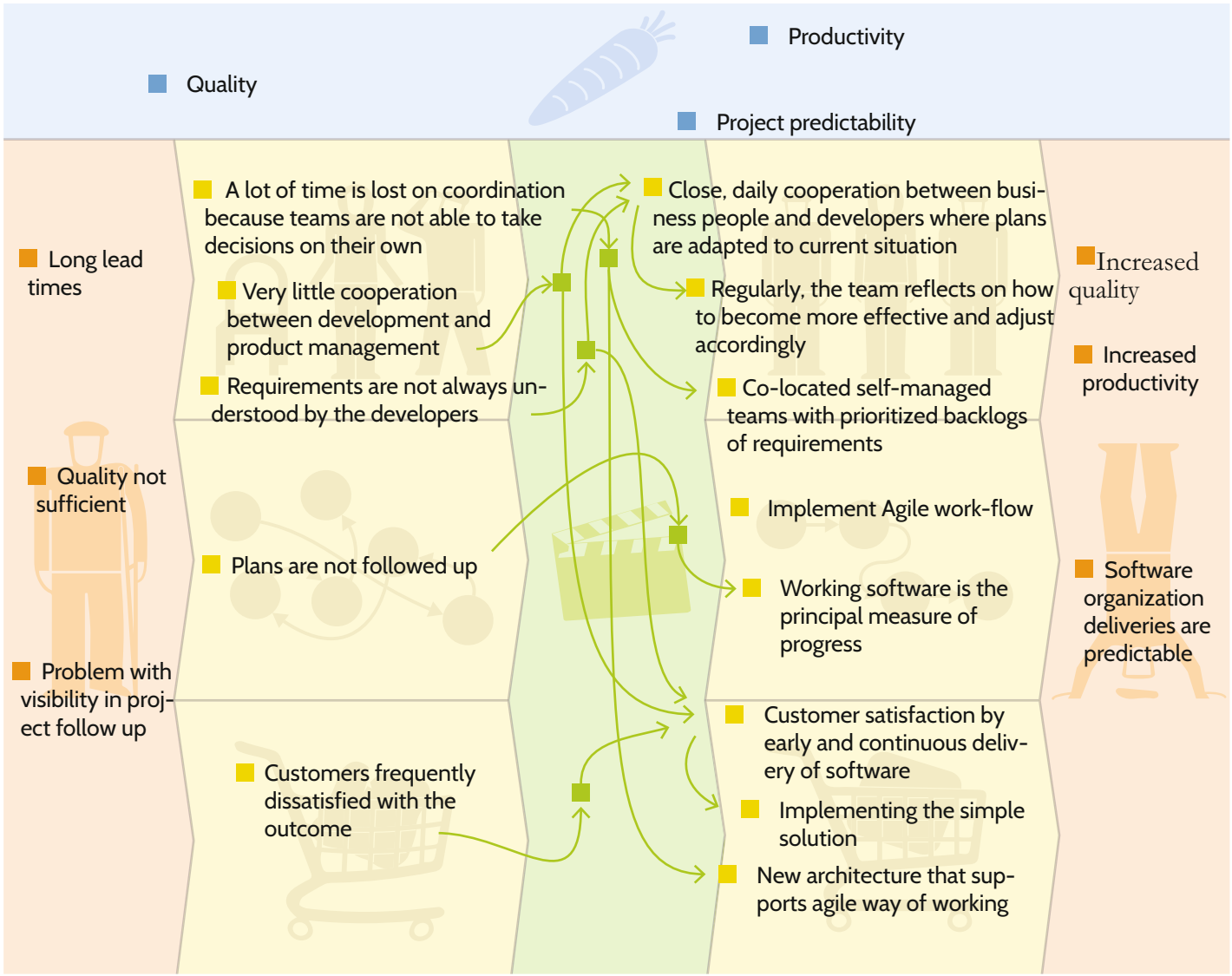
A key get-away from the project was the importance of visualizing work in progress on both project and team level, to make sure problems are detected fast. They solved it by start using JIRA* in the cloud, by this enabling smooth access by both external and in-house teams.

To split the original organization into two teams and start running Scrum took a couple of weeks. It then took almost six months to expand to five teams and another six months to become aware that the set up was not working. The resulting move from Scrum to Kanban, to get all teams and vendors on board and get everyone involved in the collaborative meetings, took another six months.

A key motivation for the change was to make their very large project finish before deadline. They desired to be able to predict what part of the scope could be completed by that time. By being able to do this, they could then re-prioritize early and solve problems as they appeared.

Did they deliver in time? Yes, they did.

* A bug tracking, issue tracking and project management tool

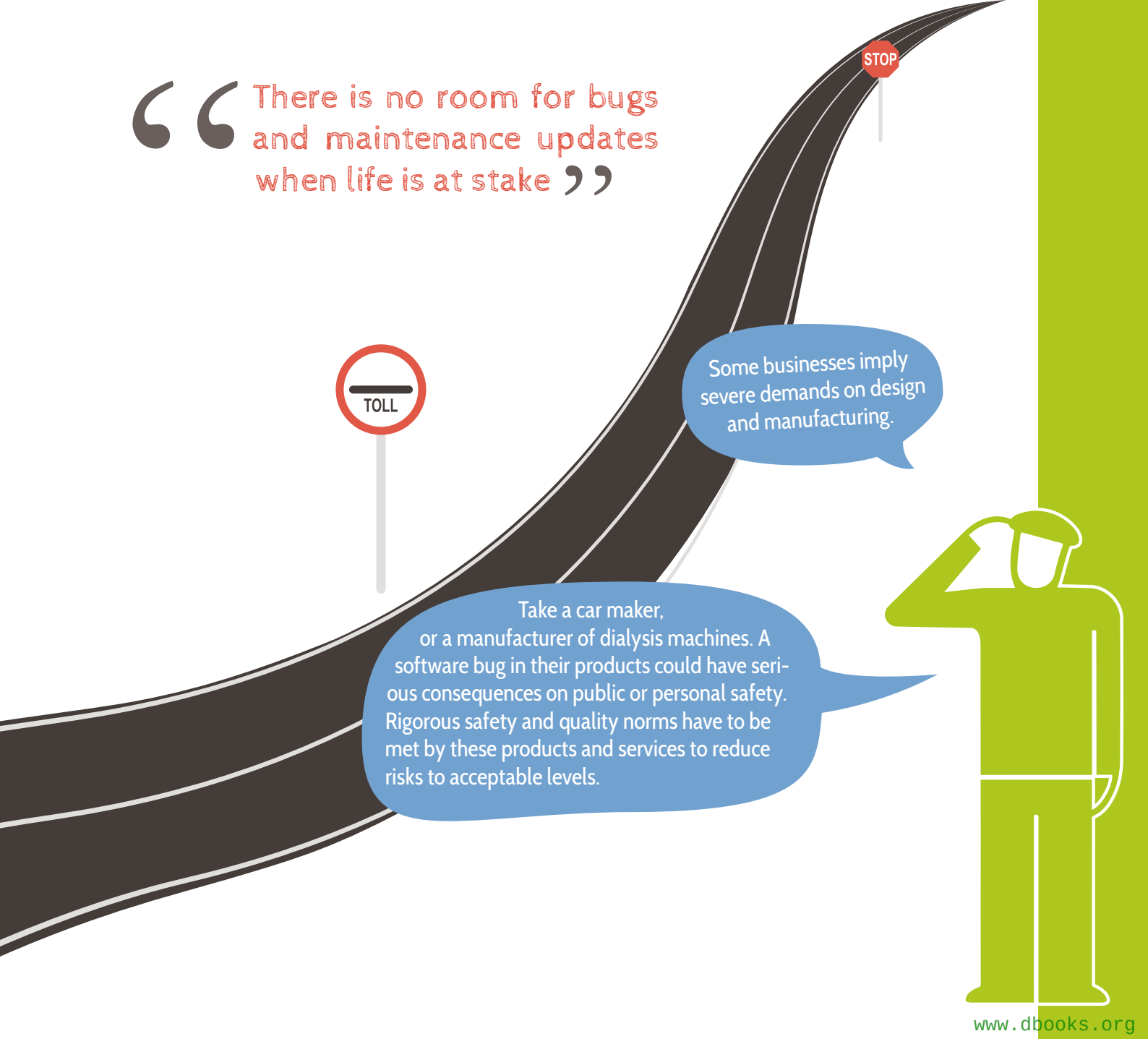


www.dbooks.org

Agile and disciplined



> There is no room for bugs and maintenance updates when life is at stake



www.dbooks.org

Regulated domains, such as automotive and healthcare, are compliance oriented. Products and services need to adhere to domains-specific standards, and so does the process of developing, manufacturing and deploying those products and services. Consider the automotive domain, for example. A car contains parts and components from thousands of suppliers. In Europe, the Original Equipment Manufacturer (OEM) takes full responsibility for the product and has to be certain that all the parts from its suppliers are compatible in performance, durability, and many other qualities attributes. All parts need to be engineered and produced according to stringent quality standards. Quality standards are not enough, though. Additional requirements such as safety and security have also been deployed as standards.



Many regulatory requirements seem to conflict with Agile software development principles. For example, the Agile Manifesto values working software over documentation—yet, it is documentation (e.g. for process traceability) that is so important in regulated domains. Therefore, it's still a challenge to apply Agile development methods within these domains.



Drivers

Agile methods have seen widespread adoption in the software industry with some surveys suggestion adoption rates of up to 80%, and for good reasons. The iterative, time-boxed approach with regular feedback cycles helps to improve software quality and customer satisfaction, and to improve developer productivity. Many of the drawbacks of traditional waterfall-based approaches can be overcome with Agile methods. However, Agile methods were initially seen as inappropriate for use in regulated domains such as the automotive industry and medical devices.

In the last few years, driven by market demands and companies' desire to improve their development processes, this assumption has been challenged, and companies in various regulated domains have explored how to adopt – and adapt – Agile methods for their specific business domain. This is also driven by trends such as digitalization in society and Internet of Things. While safety requirements are still of primary concern, even companies in these regulated domains need to consider the increasing demand for new and innovative software.

www.dbooks.org

So, the drivers that have led many companies to adopt Agile development methods also play an important role in companies that are subject to regulations and standards. They hope to achieve benefits such as quality improvements, cost reduction, and shorter time-to-market. But they also have to get better in communicating with the consumer. The digitalization trend that sweeps through our industry, and society at large, is changing customers' expectations. Customers now expect to interact with devices through web-based interfaces, and seamless interconnectivity between different devices.



Companies in regulated domains have traditionally been using waterfall-based development approaches, including the "V-model," an extension of the waterfall model, but many are now moving towards agile methods. However, while adoption of methods always requires tailoring to a specific development context, regulated domains require a number of specific considerations.

A number of general factors affect how a company should tailor agile methods. Some of those factors are:

- How many that work with the software
- Whether or not software is developed by distributed teams, and if so, how many locations are involved
- Whether or not parts of the development are outsourced
- Experiences of the workforce and organizational culture
- Complexity of the product and whether or not it concerns embedded software that runs on specific-purpose hardware
- "Greenfield"development (start on a clean slate) versus "brownfield" development (continue develop on existing software)
- Criticality of the software – whether or not the software must comply with standards and regulations

In regulatory businesses, a product has to be proven to be safe. To this end, a company can create a *Safety Case*, which consist of structured arguments supported by evidence that the system is acceptably safe for a specific application in a specific operating environment.



There are essentially five principles that summarizes safety requirements for software:

- They shall be satisfied
- They shall be maintained throughout requirement decomposition
- They shall address the software contribution to system hazards
- Hazardous behavior of the software shall be identified and mitigated
- The confidence shall be managed in relation to the system risk

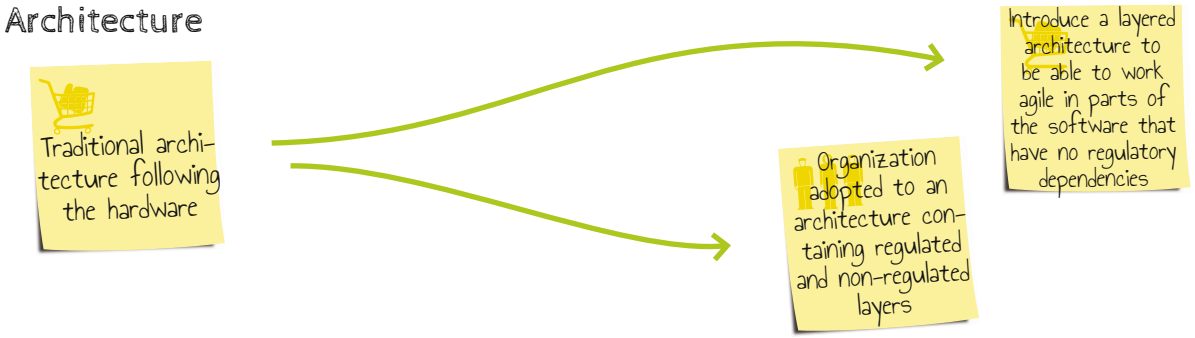
In order to provide evidence to the safety case, a few areas need to be fulfilled although they are not strict agile ways of working:

- Extensive product documentation
- Full traceability from requirements to test cases
- A documented way of working
- A documented risk management process
- Independent quality assurance

All these areas have to be adhered to satisfy the safety standards. It's not regulated how these are satisfied, but traditionally this has been accomplished by using a waterfall development method, with a heavy verification and validation phase at the end. A short-term solution of how to move away from waterfall development principles is to replace the heavy end of the project with verification and validation of releasable project increments, followed by a final but smooth validation at the end of the project. A long-term strategy requires the industry to change the standards in a more Agile direction.

www.dbooks.org

There is a short-term strategy for any company in regulated environments that wants to work agile: Apply as many Agile ideas in the development organization as possible by still following the standards that needs to be fulfilled. You wouldn't get Agile by the book, but as Agile as possible. By making the following adoptions, most companies can profit from the benefits that Agile software development has to offer.



The hardware architecture is traditionally reflected in the software architecture. This has some advantages, but also some disadvantages. It is worthwhile to split the architecture in two or more parts, where some are connected to regulatory aspects and others are not. This makes it possible to also split the organization in the same way, enabling the parts that are not affected by regulatory requirements to work in a more Agile way.

Autonomous teams



Introduce autonomous teams with full functional responsibility and by this reduce handovers and dependencies. Making decisions at the right place encourages furthermore commitment, engagement and minimizes changes and task switching.

Working code, short development cycles and continuous integration



The development work can be done in an iterative way. Create a so-called minimum viable product in weekly or biweekly steps. Always having working code increases the overall quality of the software.

www.dbooks.org

Minimum of documentation and functionality



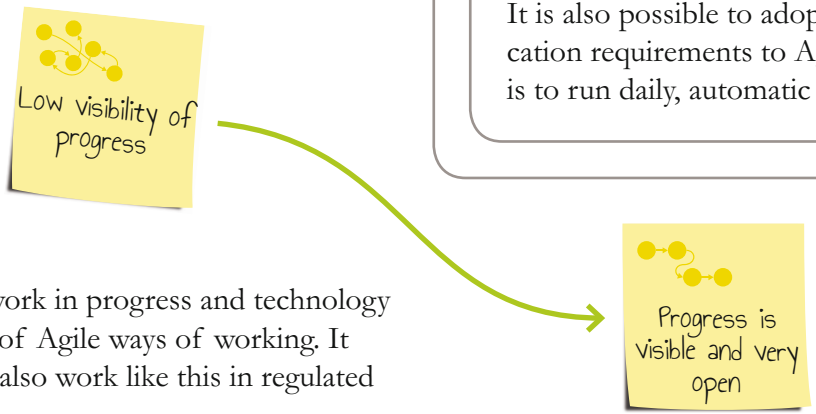
Minimizing documentation and functionality is a good strategy to increase quality. Doing as little as possible to fulfill the requirements has proven to increase customer satisfaction.

Quality and verification by agile means

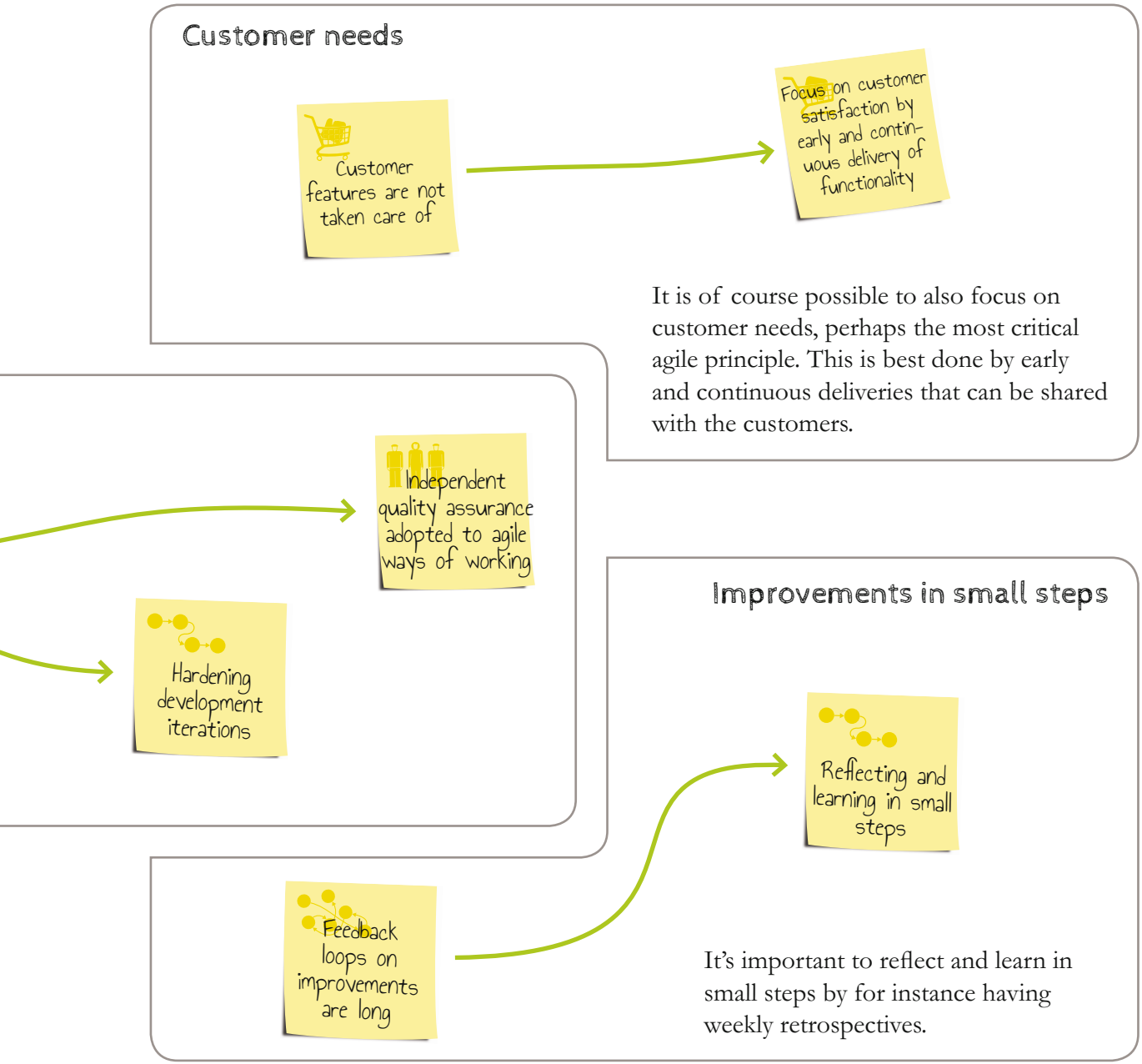


It is also possible to adopt quality and verification requirements to Agile ideas. One way is to run daily, automatic regression tests.

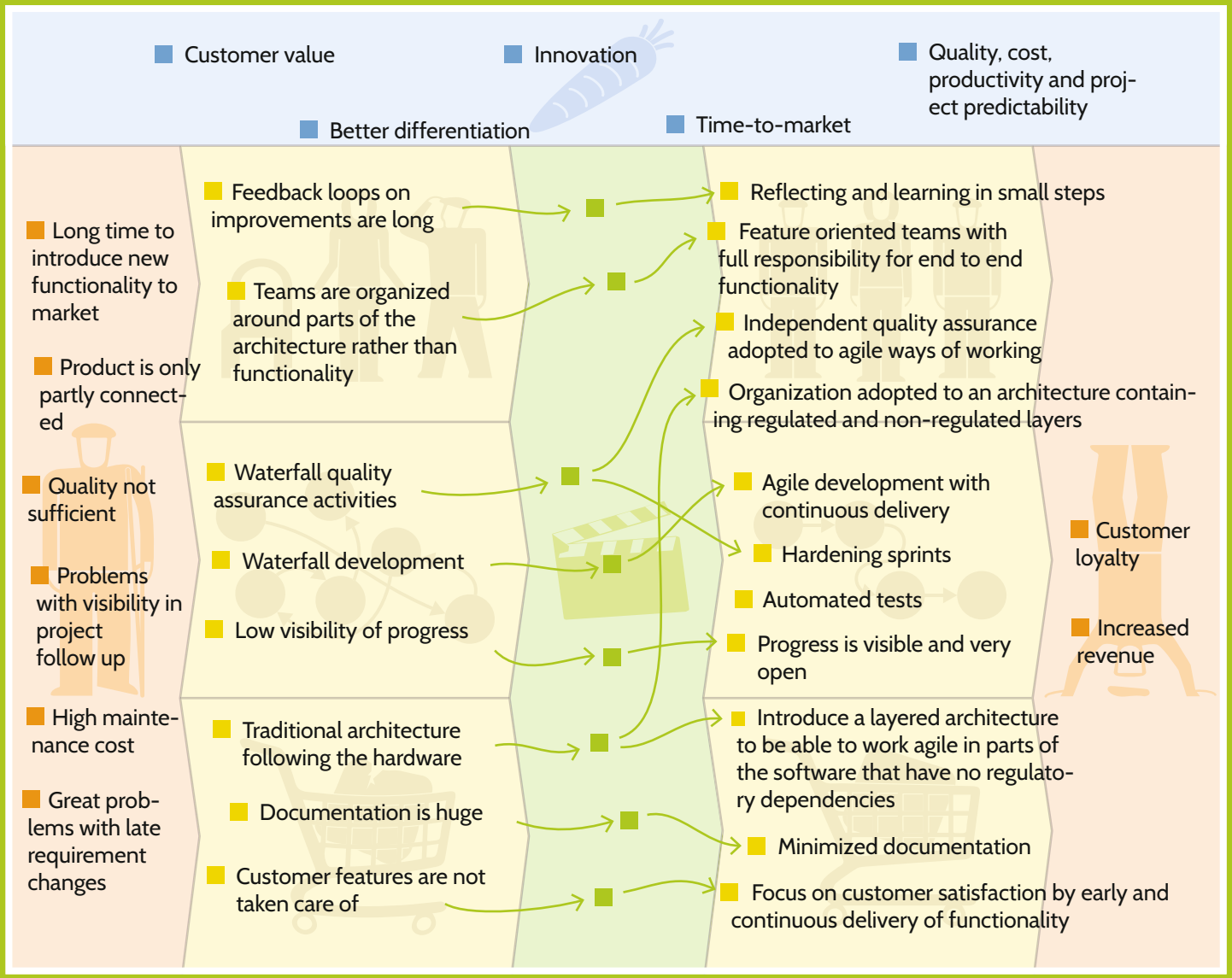
Visualized system and work in progress



To visualize work in progress and technology is a core part of Agile ways of working. It is possible to also work like this in regulated projects.



www.dbooks.org


Customer value
Innovation
Quality, cost, productivity and project predictability
Better differentiation
Time-to-market
Long time to introduce new functionality to market
Product is only partly connected
Quality not sufficient
Problems with visibility in project follow up
High maintenance cost
Great problems with late requirement changes
Feedback loops on improvements are long
Teams are organized around parts of the architecture rather than functionality
Waterfall quality assurance activities
Waterfall development
Low visibility of progress
Traditional architecture following the hardware
Documentation is huge
Customer features are not taken care of
Reflecting and learning in small steps
Feature oriented teams with full responsibility for end to end functionality
Independent quality assurance adopted to agile ways of working
Organization adopted to an architecture containing regulated and non-regulated layers
Agile development with continuous delivery
Hardening sprints
Automated tests
Progress is visible and very open
Introduce a layered architecture to be able to work agile in parts of the software that have no regulatory dependencies
Minimized documentation
Focus on customer satisfaction by early and continuous delivery of functionality
Customer loyalty
Increased revenue

Get inspired

This scenario has been based on case studies of different companies that have made this journey, to scale with Agile. Learn from their experiences, what they gained and what they had to overcome.



Scaling Agile in Automotive

Kugler-Maag are a key player who aim to bring innovations to the automotive sector, including the use of agile software development methods and open source software.

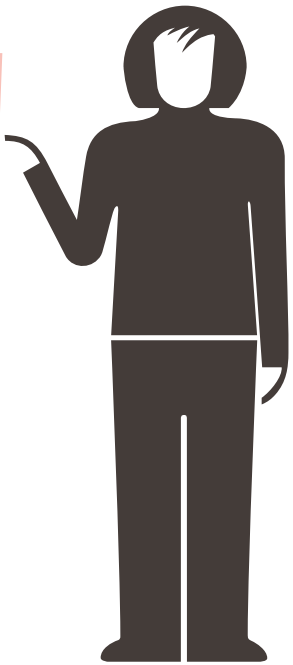


Scaling Agile in Life Sciences

Agile methods were originally considered unsuitable for regulated domains, but QUMAS found a way to scale the Scrum approach to be compliant with the standards and regulations in their domain.

One more thing

Scaling a software organization in the regulated domain seldom means to only implement Agile ways of working. Many organizations have to redesign their software architecture. Continuous delivery usually gets a high rank in the wish list. The organizations also tend to see clear competitive advantages in adapting to service-oriented business models and to work in a network of co-creators that uses open source software. All these areas have been covered by other scenarios in this book.



www.dbooks.org

Scaling Agile in Automotive



The automotive industry is going through a huge transformation. The entire industry has been disrupted by a connectivity trend. This case study is based on an industry survey conducted by Kugler Maag Cie,* a leading consulting company with many of the well-known car manufacturers as its customers. Over 40 expert interviews with decision-makers in the automotive, IT and telecommunications industries were combined with an online survey to find the main influences that affects software development. The conclusion represents the transformation that the automotive industry is in the middle of.



This study involved interviews with over 40 experts and decision-makers in the automotive, IT and telecommunications industries. A large-scale online survey was subsequently conducted to identify the key trends with respect to the role of software and its development in the automotive industry. The key finding is that, similar to many other domains, the automotive industry is currently experiencing a major transformation as the role of software is becoming increasingly important.

Customers expect their cars to be web-enabled, with many advanced features that are now custom for smartphones. Cars get increasingly more features, and similar to trends found in the smartphone industry, the car becomes a platform to which customers can seamlessly connect their peripheral devices. As a result, this increasing demand for new features and innovation delivered more quickly requires that the automotive industry responds more quickly. This is where the industry hopes the promises of agile methods can be realized. Implementing agile practices such as continuous delivery is not without its challenges, but it doesn't have to be an impossible mission.

The architecture is replaced by a layered and service-oriented architecture, containing a physical and a connected layer. This requires R&D to replace proprietary component-oriented product architectures with Internet enabling service architectures. The latter inevitable changes the R&D

* See for the full report: www.softwaredrives.com

www.dbooks.org

organizations, mostly because the culture in the organizations performing the R&D tasks for these two layers will develop differently. They have to work in a Bimodal way by focusing on speed of innovation and inter-disciplinary cooperation at the connected layer and focusing on quality and safety at the physical layer.

Also in automotive, development communities are expected to emerge dynamically around services.

The car manufacturer needs to work with open standards to quickly adjust to different organizational cultures of changing partners. In an agile organization, independent but networked units can quickly and flexibly be reconfigured, as the world changes.

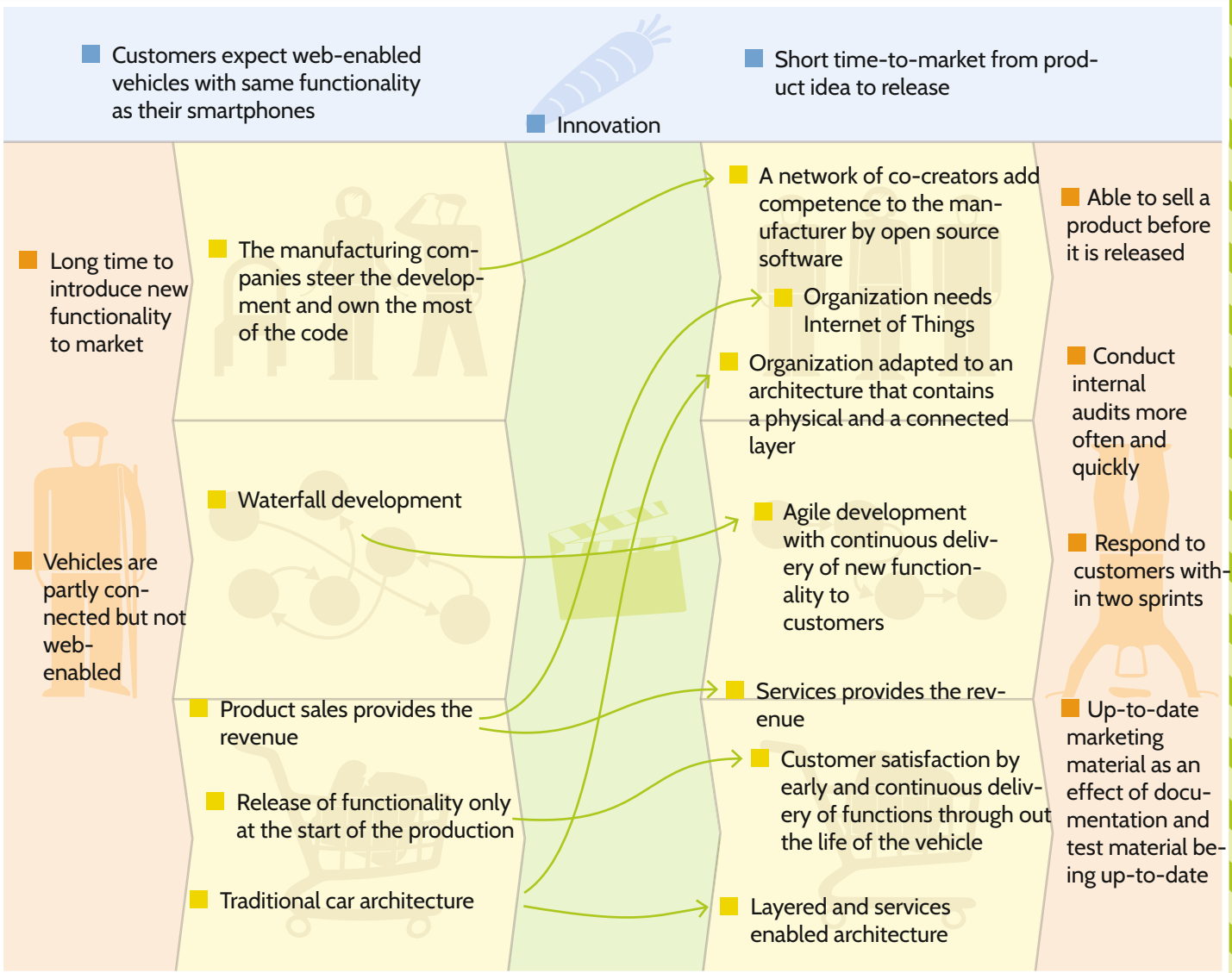
This is perhaps the most challenging part of the transformation, this that services become more important than products and only a small share of the profit is created through sales of physical products. The cultural challenges far outweigh the technological challenges.

The Internet of Things phenomenon is a critical enabler to gain more sales through service based business models. The executive management must acquire the necessary core competence to harness the emergent power of this new technology.

Once a car moves into its production phase, software development must carry on and add new functionality. Cars have to support updates and add-on apps that are developed after delivery. Naturally, the start of production-focused development has to be replaced by continuous development with short release cycles. By the architectural changes already mentioned, in combination with standardized hardware with performance reserves, the functionality of the car can be expanded significantly.

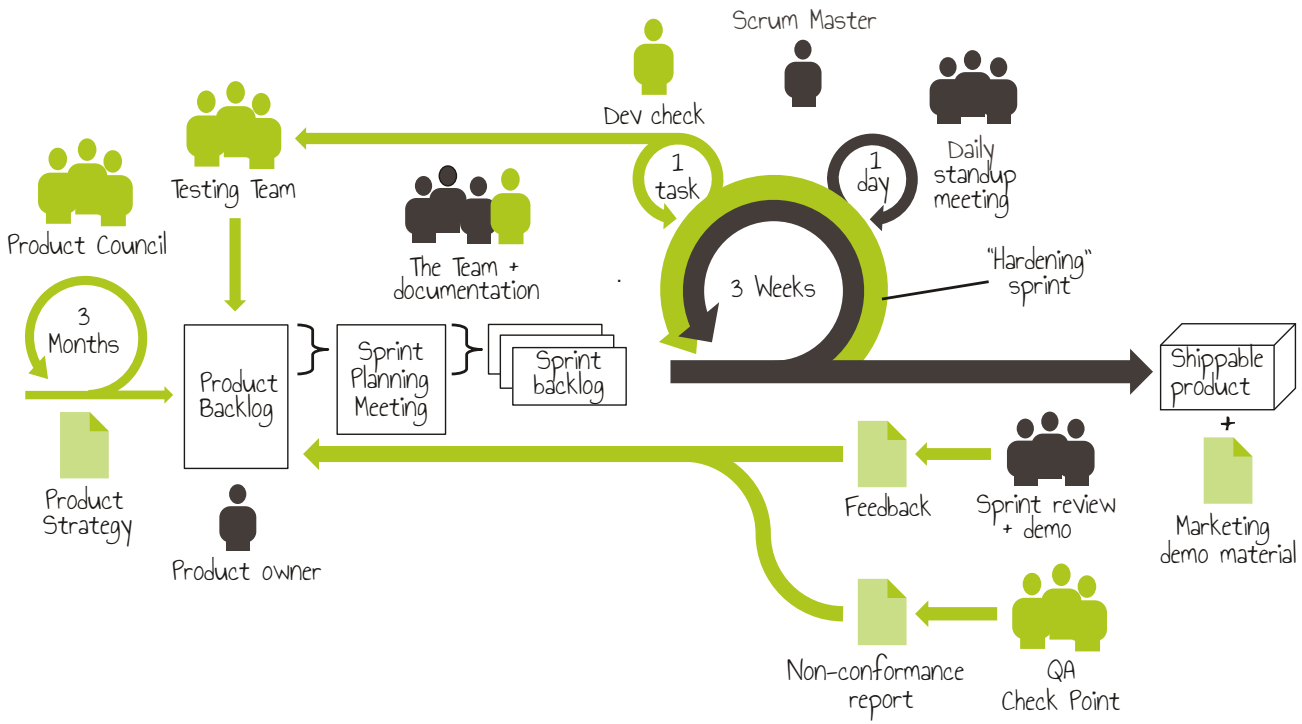
To enable fast return on investment, the organization needs to optimize the time to transit software changes to the field. It has to leverage from standards through a platform to get truly successful with continuous development, to enable additional revenue in the longer term.

Open source software in vehicles is already a reality. Open source will also become widespread in functionally critical software. This will in turn affect the organizational structure of companies as well as the way of working. The transformation has not really an end state.



www.dbooks.org

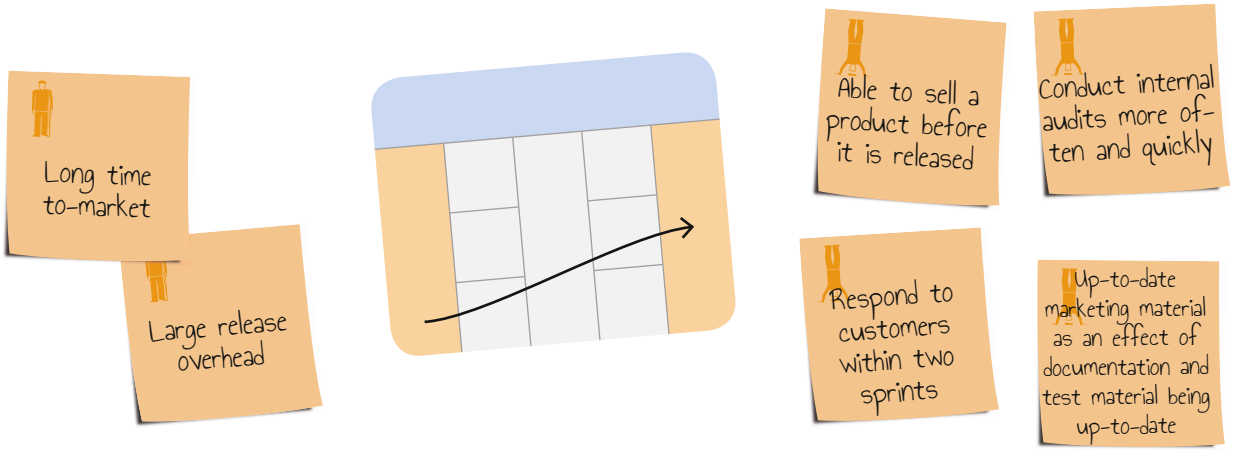
Scaling Agile in Life sciences



Agile methods have long been thought only to suit small projects with co-located teams that don't operate in regulated domains such as the automotive and medical sectors. This case study describes how QUMAS, a leading supplier of regulatory compliance management software to the life sciences sector has tailored the standard Scrum framework to regulated environments.



QUMAS had employed a classic Waterfall approach ever since the company was founded. The approach resulted however in a long time-to-market and a large release overhead, all-in-all quite serious weaknesses on a rapidly changing market such as the one QUMAS operates in. To combat this, the company spent about two years to adopt and augment the Scrum methodology.



www.dbooks.org

The Agile Manifesto offers four value propositions:

Individuals and interactions over **Processes and tools**

Working software over **Comprehensive documentation**

Customer collaboration over **Contract negotiation**

Responding to change over **Following a plan**

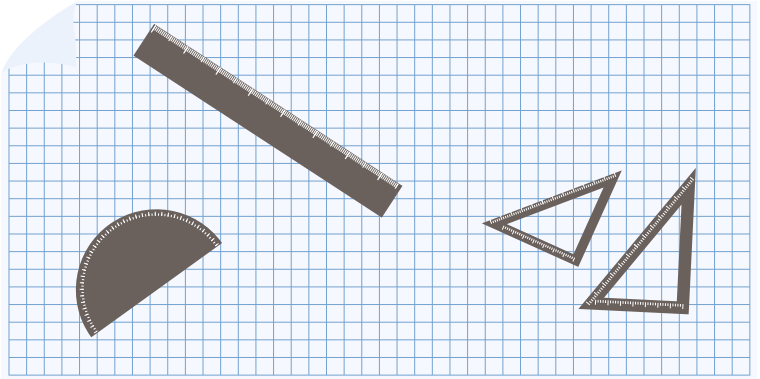
While agile advocates accept that the blue statements on the right are important, they value the red statements on the left more. However, in regulated environments, the blue statements on the right loom very large and are perceived to be key. If it isn't documented, it isn't done is a frequent refrain in the regulated domain. Agile methods may for that reason appear to be inappropriate for regulated domains. They aren't, of course, but they need to be tweaked to fit the regulatory requirements.

The standard Scrum framework defines roles, ceremonies and artifacts. The product owner, the team, and the scrum master are for instance roles. The ceremonies include activities such as the daily stand-up, the sprint planning meeting, the sprint review and the retrospective meeting. Artifacts include the product backlog, the sprint backlog and at the end of every sprint, a "shippable" product. QUMAS's development process is regularly audited. In order to comply with the various regulations they are subject to, QUMAS has extended the standard Scrum framework with a number of additional roles, ceremonies and artifacts, resulting in R-Scrum: Scrum for Regulated domains.

New roles	New Ceremonies	New Artifacts
Quality Assurance	Dev Check	Marketing demo material
User documentation	QA Check Point	Updated design documentation
	Hardening sprint	Non-conformance report

Quality Assurance is an important additional role in R-Scrum. Regulations require that QA is independent from the development team. The QA Check Point is a new ceremony that takes place after every sprint, when QA conduct an internal audit to ensure "continuous compliance". Rather than conducting an extensive, annual audit, audits now take place after every sprint. Any issues that emerge during the audit are reported in a non-conformance report, which is addressed in the next sprint.

The user documentation role is also new and assigned to at least one member of the development team. The team has also got a new ceremony, the Dev Check. After a task is completed, any code and documentation is peer reviewed by another developer. This is required by the regulations that QUMAS must adhere to. Another new ceremony, the hardening sprint, should be done just before the final release to ensure that the product is fully compliant with all necessary regulations.



Traceability is a key concern in regulated domains. QUMAS have adopted the Atlassian toolset, which offers full end-to-end traceability. Jira is used for issue tracking and project management. Other tools in the toolset offer source code search, an enterprise wiki, agile planning and project management, continuous integration, and peer review. The toolset is a key ingredient to the Agile transformation and facilitates a very effective audit process.

www.dbooks.org

Lessons Learned

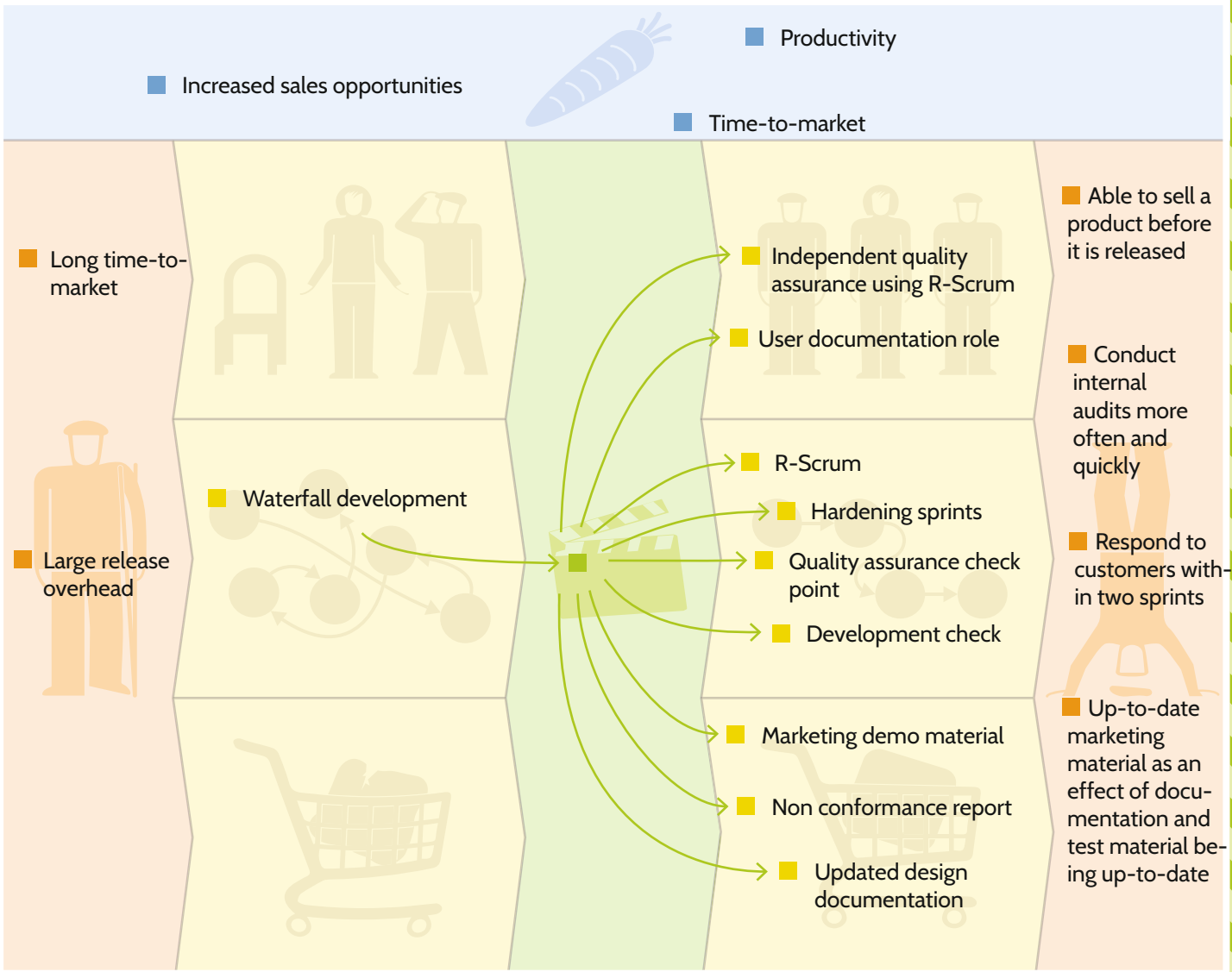
QUMAS have successfully tailored the standard Scrum framework to facilitate the additional constraints imposed by the regulations that their development process must consider. The key lessons learned of this case study are:

- A fully integrated toolset is essential to support the R-Scrum method and to ensure "living traceability."
- It is necessary that the QA department also adapts; the migration from a waterfall process to a Scrum process cannot be done without organizational changes. QA must also adapt to a "sprint schedule" in order to achieve "continuous compliance."
- Additional roles and ceremonies such as the Dev Check and the user documentation role are needed to ensure that any code that is checked in is compliant and properly documented.

The sprint-based approach allows QUMAS to always be able to demonstrate the latest version to potential customers. The marketing demonstration material is always up to date. This has greatly improved QUMAS' sales opportunities. Based on product demonstrations, several customers have pre-ordered new products, even when they were still under development. This by itself is noteworthy, and is untypical of in regulated domains.

The "scaling" of QUMAS' development process has also had a significant impact on the organizational and the product domains. QUMAS' story is therefore representative of the "scaling software" phenomenon that this book focuses on.

.



www.dbooks.org

Outside the box



Over the years,
focus in outsourcing and offshoring has gradually shifted from low-cost to factors such as qualified personnel, ability to ramp resources and access to an international market. To solely focus on low-cost has turned out to be counter-productive.

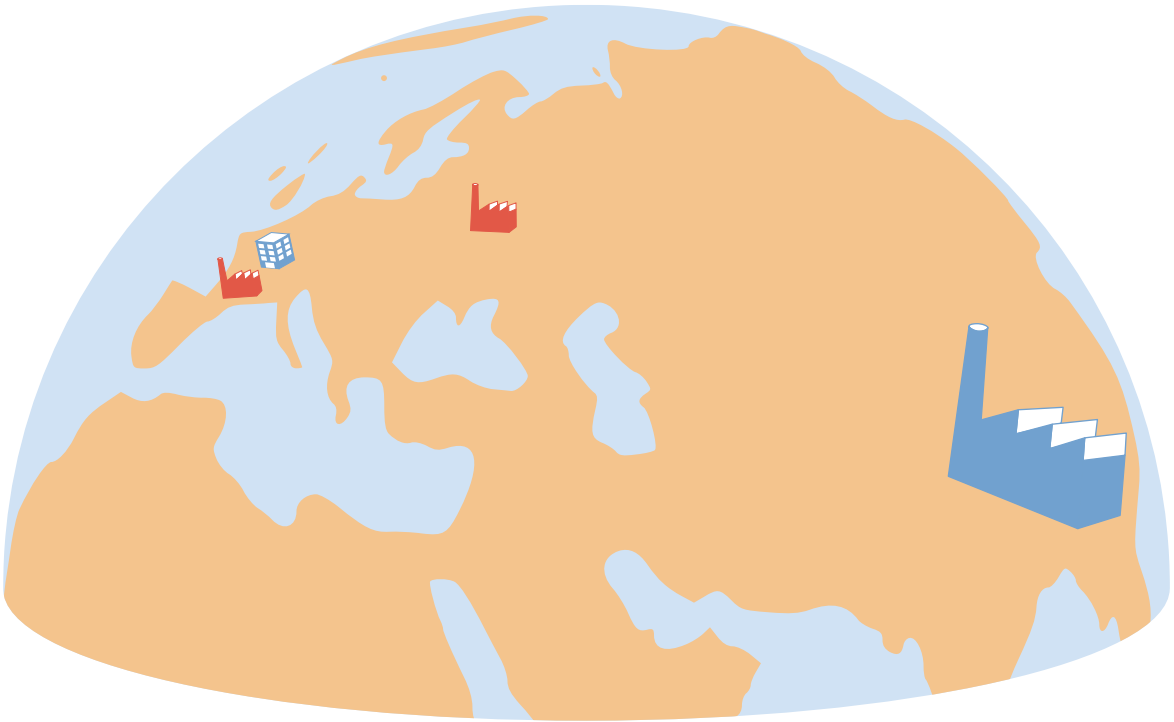


www.dbooks.org

Even if development cost reduction is the most common reason for choosing Outsourcing or Offshoring as a software development strategy, there are many other reasons why companies embark on such an endeavor. A bit of clarity over the terms will shed light on what these other benefits might be:

Outsourcing means that we contract a third party to develop what we used to develop ourselves. They can very well be located in the same town as we are.

Offshoring means that we relocate all or part of our development to another country. We are still doing the development; we're just doing it abroad.



Putting the low-cost aspect aside, managing workload peaks is a good reason to choose outsourcing, as it is costly and risky to build up and maintain internal overcapacity. Further, to balance our investments and risks, it also makes sense to have our own developers working on the most important areas and letting a third party take care of less critical development. Another benefit that outsourcing brings is agility in scaling development capabilities (both increasing and reducing) and therefore ability to faster react to market and business fluctuations.



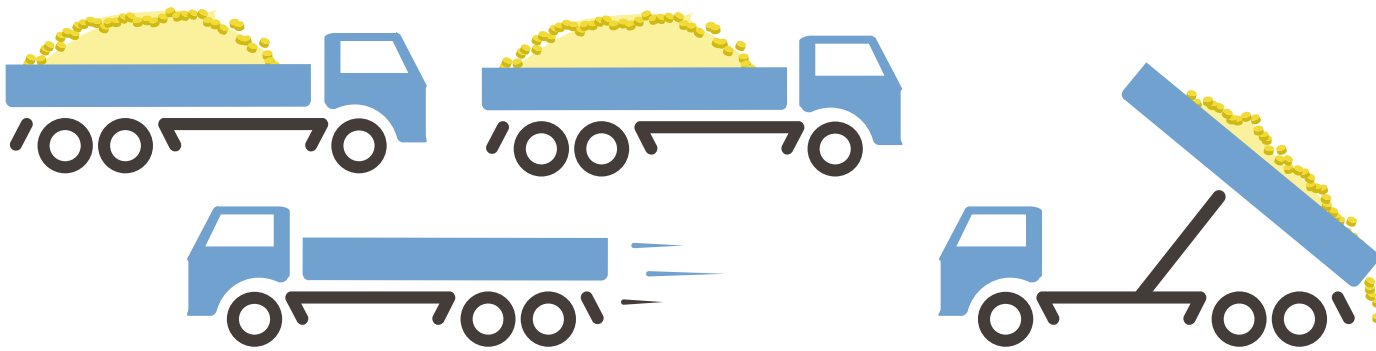
The reasons for choosing offshoring are similar as those for outsourcing. Another good reason is the continuous character of an offshoring relationship, with permanent cost reductions and without the hassle of contract negotiations with vendors. Moreover, it enables organic knowledge transfer, growth and "follow-the-sun development."

www.dbooks.org

Many companies choose near-offshoring where time zone differences are minimal and travel time is short between locations. Other companies decide to offshore to regions very far and with time-zone differences of six hours or more.

It's really a matter of how far we want to take it. It's generally more difficult the longer we get and the more we detach. But it's not impossible. So, bear with us while we outline a strategy that can get you where you want.

The unforeseen costs



Over the years, the trend in outsourcing has gradually shifted focus from reducing costs to factors such as availability of qualified personnel, ability to ramp resources, and access to an international market. Focusing solely on cost reduction has turned out to be contra-productive.

But still, it's ever so common that an outsourcing project starts with a clear drive to cut costs, only to turn around halfway moving into firefighting mode. If not well prepared and managed, a project can become more expensive than when the outsourced task was done internally. And worse,

quality issues may negatively impact customer satisfaction and drive the sales down the drain.

So, we need to get aware of what the real costs are. Too often, the cost and investment calculation are based on a price per person-hour comparison, which hardly give the full picture of the total cost.





We can expect additional costs due to:

- Time and effort to transfer knowledge and projects to a third party.
- Initial quality and security concerns.
- Delays and long lead times due to communication issues, cultural differences and geographical distances.
- Lack of competence and training -- the supplier may not have the same competence and the ability to achieve the same velocity in development and thus compensates by adding more resources than needed internally.

This might come without saying, but better safe than sorry: we should expect the running costs for managing the outsourcing partner, requirements and deliveries to be higher compared to if we had continued running the project internally.

Staying in control over the costs, or investments which they rather are, will help us not making hasty decisions when taking on the real challenges.

www.dbooks.org

We're only human

There will be challenges. Certainly, there are matters we can't predict and matters that are totally out of our control. But many problems that arise in an outsourcing or offshoring project can be traced down to human nature. It's due to how we communicate, how we motivate and are motivated, and make everybody believe in the project (or not), how we make the journey inclusive, and a million other things. We have to deal with them and take these issues seriously.

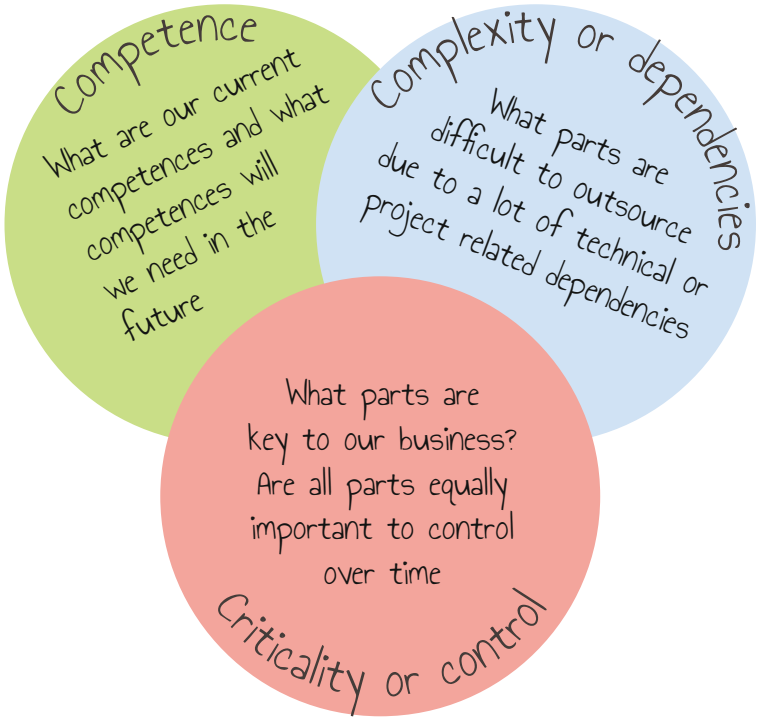


On our home site, employees will worry about their jobs, even be annoyed over the idea of having to train and transfer their knowledge and work to those that replace them. They will make assumptions, rightly or not, leading to an inner resistance to cooperate. Another challenge is the need to change roles, routines and processes for the staff remaining at the home site. It's not unusual that completely new skills and competences are required, to support an efficient global delivery set-up.

At an external site in for instance India or China, we should expect it's a challenge to attract and recruit highly skilled engineers. Most outsourcing partners have great difficulties in retaining their staff, resulting in a very high employee turnover. The competition between global engineering companies that dip in the same pool of skilled workers is fierce. Indeed, there are more skilled engineers available than ever on the market, it's just very difficult to hire the best. There are indeed lots of engineers with knowledge in modern, standard based technologies. But if our system is built on old technologies or requires specific knowledge, they get fewer. This has been analyzed thoroughly in this and other studies.

Due diligence

Enough about difficulties, let's get down to business. To succeed with a mission such as this, we need to analyze and prioritize the following parameters:



Given the costs, distractions, investment of management time and other hurdles that will come when getting into outsourcing or offshoring, the importance of the due diligence can't be emphasized enough.

www.dbooks.org

Our strategy

Once having the due diligence done, we're ready to outline our outsourcing or offshoring strategy:

Why do we want to outsource, what are our goals? Is the outcome measurable?

It is very critical to set clear goals and expectations because it shapes everything that follows from this point. Without clear goals, we can foresee unfulfilled expectations and uncertainty about the outcome of the entire operation. How would we know if we have succeeded? So, depending on what we want to achieve, what our business drivers are, we could for instance elaborate with the following targets:

1. Outsourcing ratio (e.g. 80% of our staff are outsourced and 20% of our staff are in-house)
2. Cost-saving (e.g. we cut 50% from the current cost)
3. Market presence (e.g. 10% of the market share in a partner's country or region).

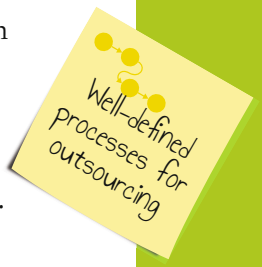
What do we want to outsource?

We need to make a make-versus-buy analysis to determine what's possible to outsource and what's not. For instance, complex parts that aren't easily decoupled or used by several other projects won't likely be easy to outsource. On the other hand, we might actually want to outsource a part even if it would be cheaper to develop it ourselves. This would for instance be the case if we want to allocate our own resources to more critical activities or if we want to build up and maintain capacity for peaks in our workload. It all depends on our goals.

Outsourced components are chosen from the use of a Make-Buy strategy

Who can we outsource to?

In addition to cost, we need to analyze the third party supplier's competence and maturity, development method and delivery model, their financial, legal and security status. We need to understand the political situation, and geographical and cultural contexts. If our development team will be tied up in ongoing projects and our deadline is tight, we could consider bringing in consultancy help. Having clear goals is key to be able to objectively select a partner.



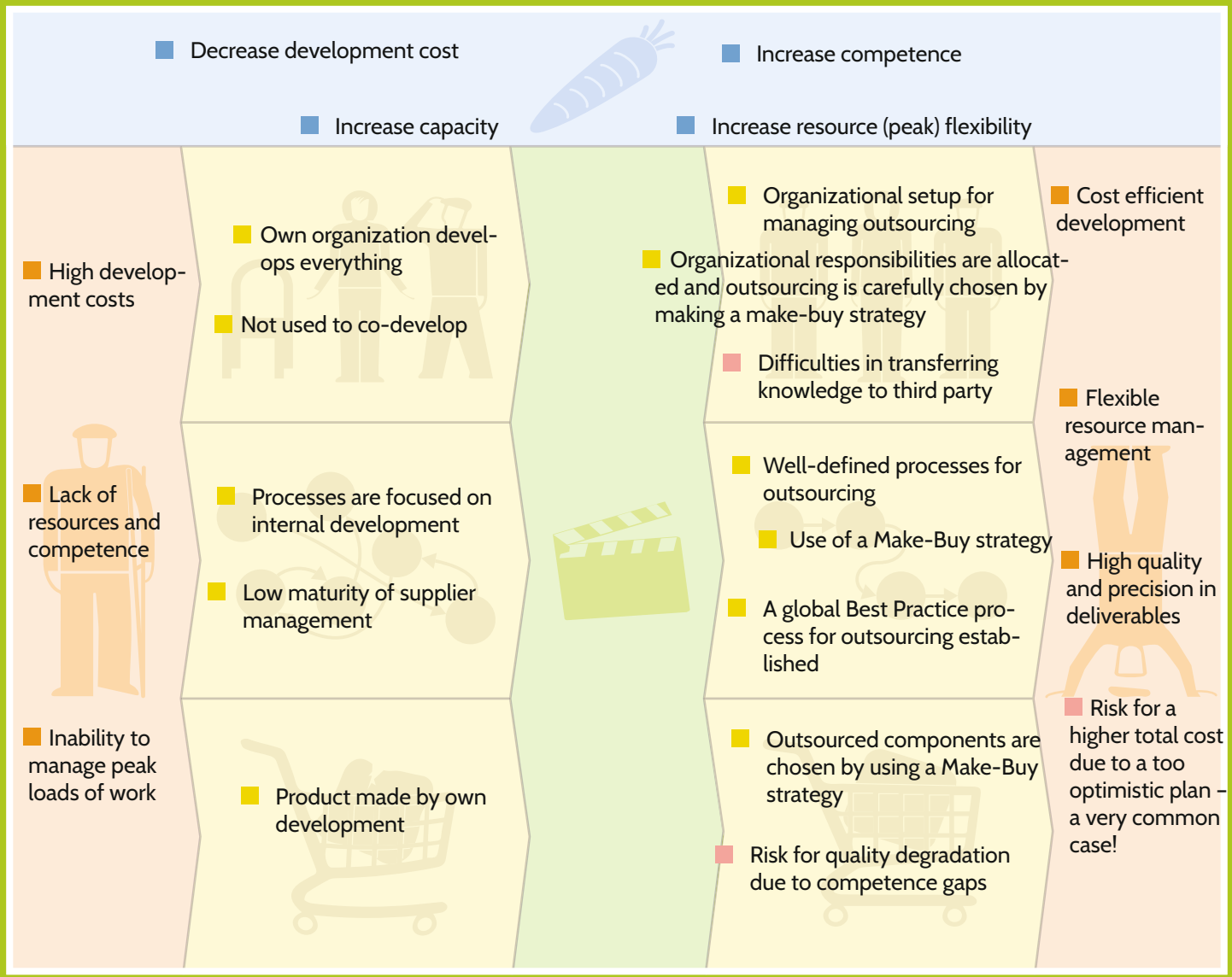
It is critical to stay on top of things and be prepared when the transformation doesn't run as smooth as anticipated. The goals are the cornerstones in the measurements and the quality assurance we need to put in place. Both direct and indirect costs need to be taken into account. Proactive decisions need to be made at the first sign of discrepancies between plan and reality. To succeed in this transformation, we need to be agile and flexible.

How can we organize ourselves?

Which new roles will be needed to manage the supplier and their deliverables? While setting up an organization isn't that complicated, transferring necessary product and process knowledge requires substantial effort and time. Not only do we have to make several trips to the supplier's location, staff from both sites will have to meet face to face, not only to get to know one another, but more importantly to better understand the tasks and challenges they face. It is particularly important to introduce incentives for the employees at the outsourcing site in order to minimize staff turnover.






Decrease development cost
Increase competence
Increase capacity
Increase resource (peak) flexibility
High development costs
Own organization develops everything
Not used to co-develop
Organizational setup for managing outsourcing
Organizational responsibilities are allocated and outsourcing is carefully chosen by making a make-buy strategy
Difficulties in transferring knowledge to third party
Cost efficient development
Lack of resources and competence
Processes are focused on internal development
Low maturity of supplier management
Well-defined processes for outsourcing
Use of a Make-Buy strategy
A global Best Practice process for outsourcing established
Flexible resource management
High quality and precision in deliverables
Inability to manage peak loads of work
Product made by own development
Outsourced components are chosen by using a Make-Buy strategy
Risk for quality degradation due to competence gaps
Risk for a higher total cost due to a too optimistic plan – a very common case!

Get inspired

This scenario has been based on case studies of different companies that have made this journey, to scale with Offshoring or Outsourcing. Learn from their experiences, what they gained and what they had to overcome.



Efficient communication in a global delivery model

Since this scenario is about the glam of succeeding, and not the gloom of struggling, do pay attention to the case study of Tieto.



Outsourcing Strategy at Sony Mobile

Read about the smartphone manufacturer, which journey started like many others at the time, a bit on a bumpy road. Eventually, they scaled into an organization that was able to identify parts that are best suited for outsourcing, to continuously introduce and manage outsourced development projects along with their internal development.



Not so shore anymore

This company had an equally bumpy experience when they decided to outsource a system that was not particular well suited for the purpose. It didn't go well, but there are still many lessons to learn from their case.

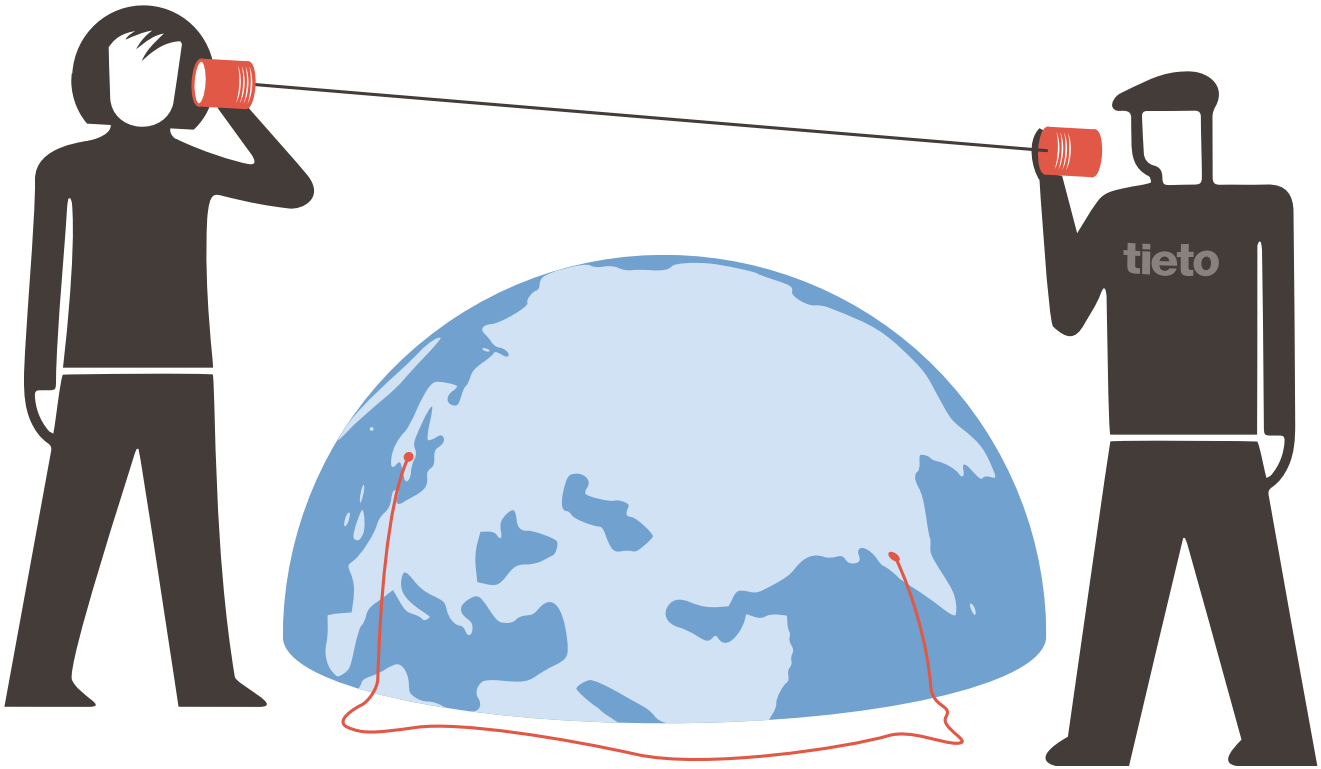


Play it again, Sam, backwards

Another story to learn from is the rise and fall of Sony Ericsson's PlayNow service. It's an excellent example of when bringing development back and run it in-house makes sense. They should not have outsourced in the first place.

www.dbooks.org

Efficient communication

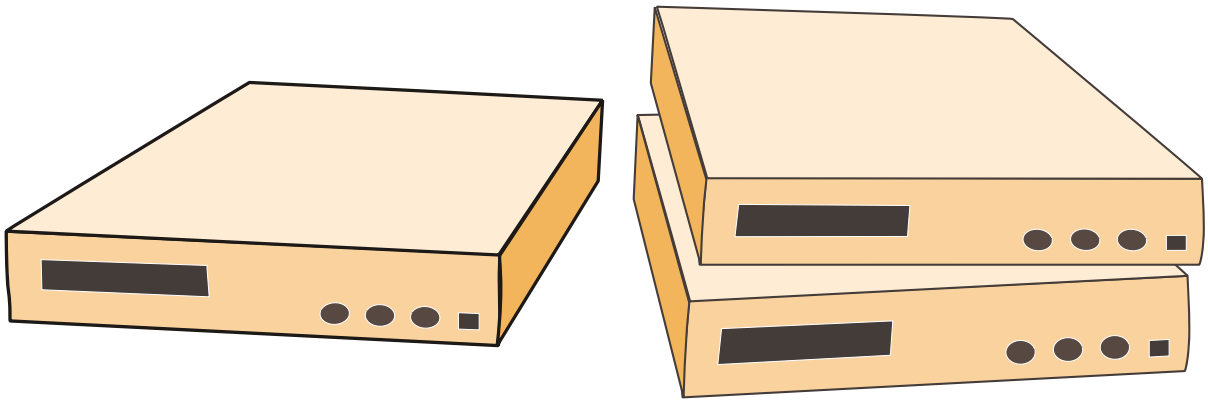


in a global delivery model

This case study features Tieto, one of the largest IT suppliers in the Nordic countries. As Tieto operates in many different locations and with different suppliers, which is why they seek ways for efficient communication in a global delivery model. The main driver for offshoring was to reduce costs and release personnel for design of the next generation of systems, and at the same time to maintain high system availability and service levels.

The system we focused on has been in operation for over 20 years and is classified as very complex with a large number of integrations, databases and functional modules. The system is business critical and used in the daily work by approximately 3,500 users.

To reach a more cost efficient set-up, a major part of the delivery was transferred to a Tieto offshore site in Pune, India in 2010. The goal was to reach an offshore ratio of 80% and to keep a team with strategic architectural competence in Sweden.



www.dbooks.org

The case study was performed five years after the transformation and conducted through several interviews with persons involved both before and during the transformation. As the level of the offshored activities increased, also the need for project communication and knowledge transfer increased. These two factors were later identified as being the key success factors in this operation.

The focus of the case study has for that reason been the importance of good communication and knowledge transfer in terms of:

- Competence
- Processes
- Organization
- Requirement handling
- Motivation and engagement
- How further improvements can be made

The first step in the transformation was to appoint and hire resources in India and send them to Sweden for an 18 weeks training program. Next, senior architects from Sweden were sent to India for 6 months to build up the competence level of staff of the Indian team. To assist training, specifically designed online courses were offered which was an efficient approach. The Indian team gradually increased their system knowledge and could take over responsibility for more complex tasks already during the transformation period. The regular visits have continued after the transformation, in both directions.

The goal was to establish "one team" distributed over the two sites. Instead of creating a process where each site was responsible for different phases and deliverables, a single team was built based on the roles that were needed. The purpose was to bridge between the sites and get an efficient communication and knowledge transfer within the team.

Observations



Ad-hoc peer-to-peer communication proved to be very efficient in the previous, single site organization. Splitting the team over two locations resulted in a more complex communication structure between engineers from different cultures and locations. The team now needed communication solutions such as chat, voice and video conferencing.



The complexity of the system functionality was also problematic. It took longer for the offsite part of the team to learn and understand the solution functionality than the technical solution. The implementation techniques were often quite generic. Most difficult to learn was however the customer-specific requirements and processes and the various difficult legal constraints.



Both the Swedish and Indian teams were exposed to new cultures. There are significant differences in the way of working, communicating and collaborating. The Swedes were surprised about the extensive hierarchical approach to responsibilities and roles that were common practice in India. This allowed the Indians to create an escalation hierarchy that significantly saved time among the Swedish architects.



Another significant cultural difference was high personnel turnover in the Indian team. To switch employer is common and a cultural norm in India. This turnover requires additional training efforts and jeopardizes successful and sustainable knowledge transfer.



The case study team also noted that communication efficiency appeared to be much higher after visits between the two sites. In addition, a business trip to Sweden was regarded as a very attractive goal in itself and highly motivating for the Indian team.

www.dbooks.org

Recommendation on how to succeed

Make a thorough analysis of the system before selecting competence needs and communication strategies. It is more challenging to get an effective offshore for complex systems, so focus on functional complexity rather than technical solutions.

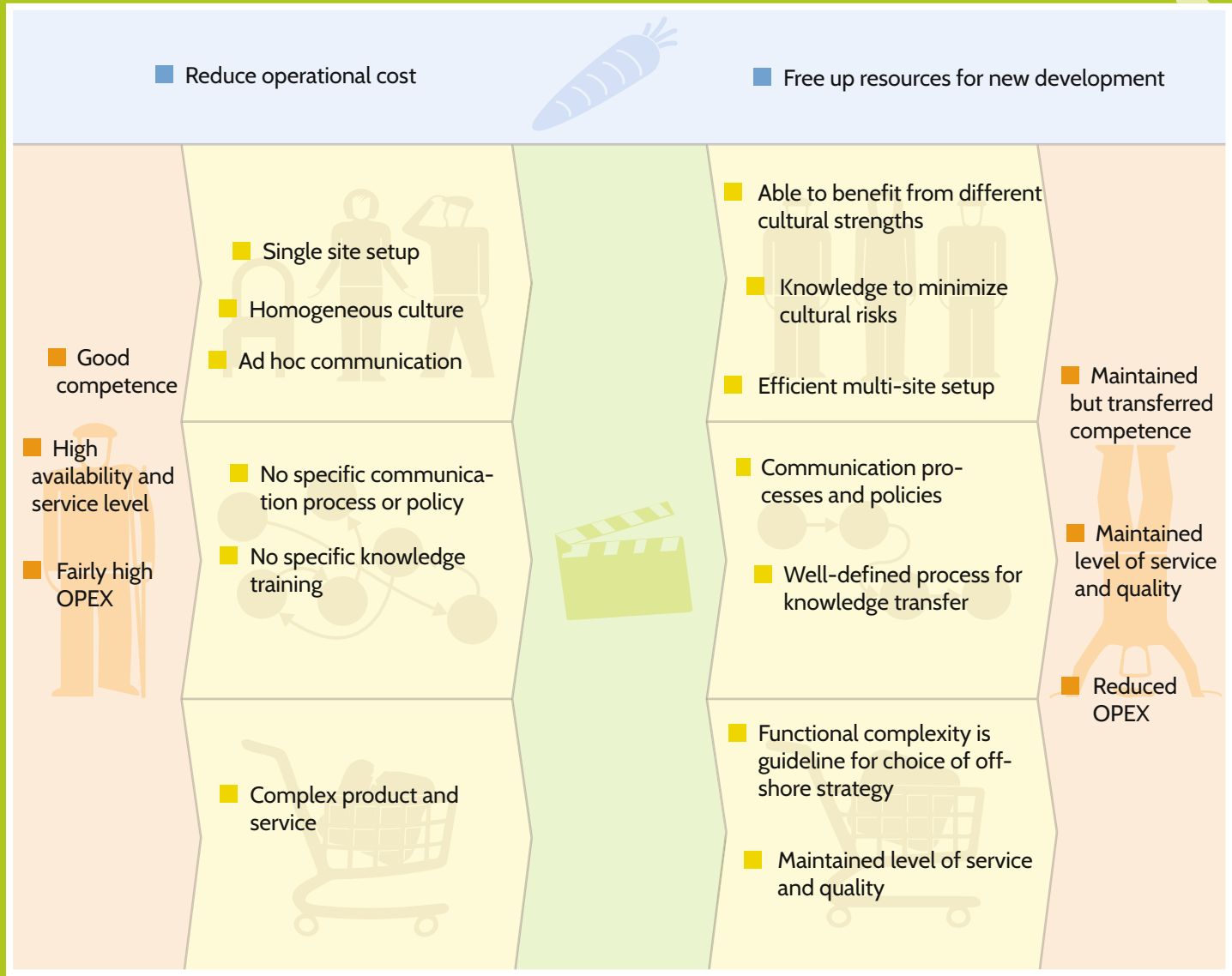
Decide upon a competence strategy early in the offshoring phase. Take this into account when staff reduction starts at the local site. It's important to secure personnel for future roles available in later phases of offshoring.

Make a visible step-by-step knowledge transfer process. Tailor training programs to areas of expertise and let team members mature over time, area-by-area. Repeat steps per area:

1) Self-study using existing training material (docs and videos)
2) Supervised trial operative work
3) On-site training with architects
4) Unsupervised operative work

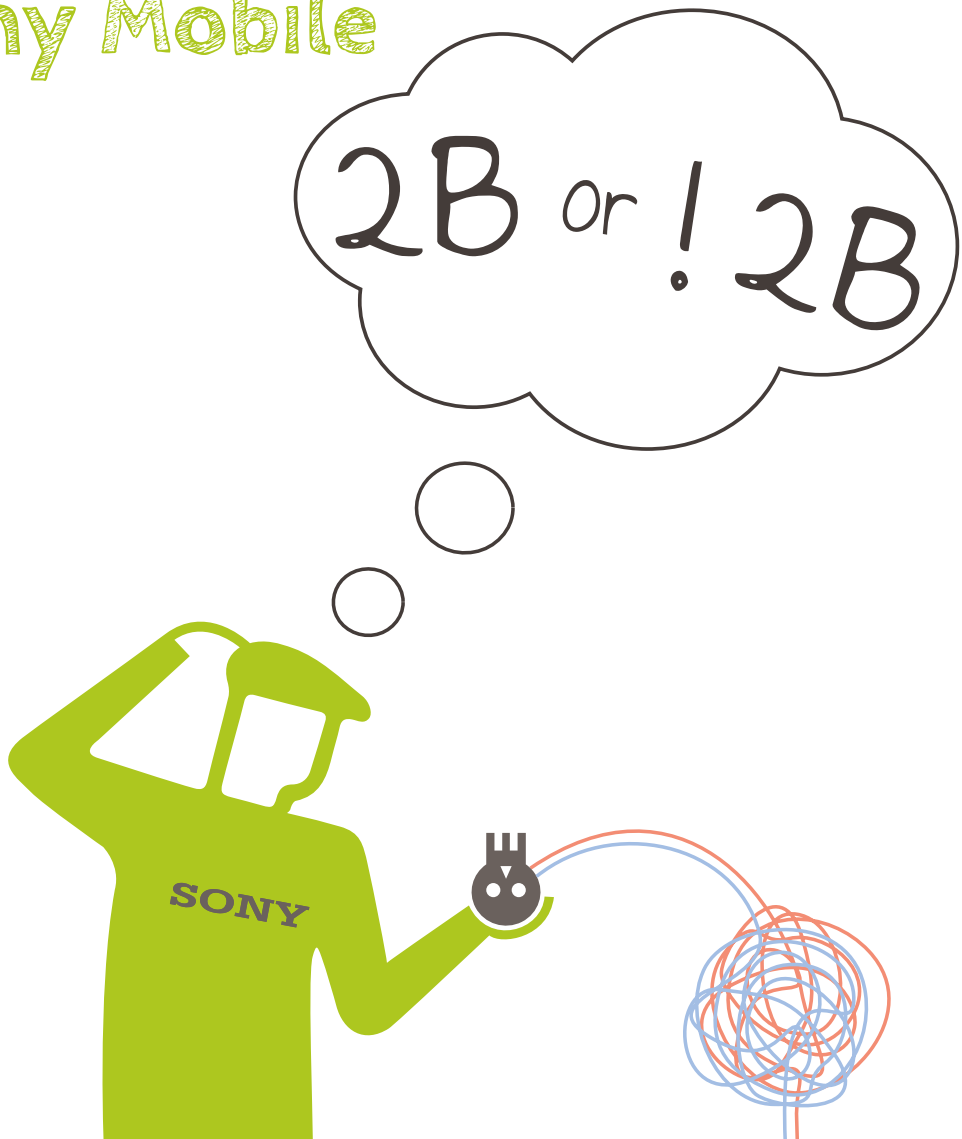
Do not underestimate the difficulty to achieve good awareness in the project, to ensure that all team members know what is going on.

- Make an analysis of which recurring meetings that are needed and decide on frequency, participants, purpose and scope.
- Decide on communication channels to use and establish good conducts.
- Reduce the amount of redundant communication. Rules and processes for how to communicate can solve this problem. Be careful, however, as it also creates latency in communication and reduced awareness between sites. Documentation solutions like the wiki will also help reduce redundant questions.
- Create processes to continuously secure the quality of the documentation. Old information must be removed and relevant information must be searchable and readable.



www.dbooks.org

Outsourcing Strategy at Sony Mobile





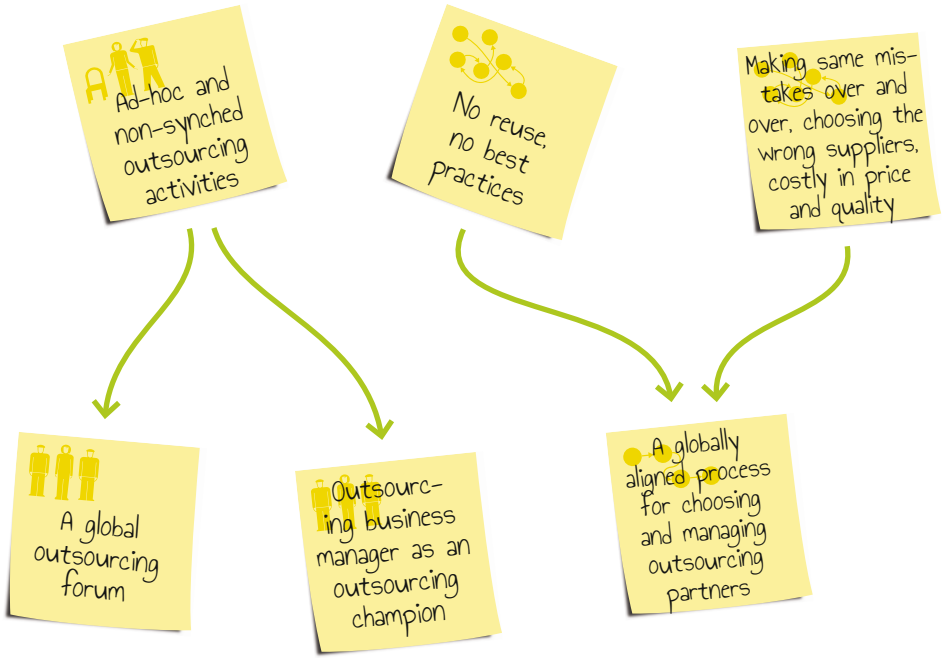
Software has transformed the mobile phone industry. Whereas the first mobile phones contained a minimal amount of software, today mobile phones contain more powerful processors than those used to put man on the moon. This allows modern phones to do much more than just making phone calls, offering many more advanced features. To develop software that makes this possible, all major players in this industry have outsourced some of their software development – and Sony Mobile is no exception. For Sony Mobile, the main driver was to reduce development costs, sustaining growth, and to mitigate difficulties in finding well trained developers.

All these reasons are very common throughout the software industry. Unfortunately, not many companies perform a thorough analysis to evaluate whether cost savings are realistic and achievable. Sony Mobile didn't stick out here either. Too often, companies embark on outsourcing journeys solely to reduce costs based only on a simplistic comparison of the hourly wages of developers. This, however, leads to a completely wrong conclusion when other factors are not included in such calculations. When starting on an outsourcing journey, companies need to spend considerable efforts and expenses on knowledge transfer activities, onboarding, and companies must also anticipate various barriers that might emerge due to more complicated communication that is now hindered by time zones and geographical distance.

Outsourcing partners – the supplier that will do the customer's work – often don't possess the same level of knowledge and experience as the customer company, and often this lack of knowledge is compensated by adding more people to a the project, all of whom take considerable time to build up domain knowledge. This is the first way in which the total cost might end up higher than anticipated – perhaps more than if the software were developed in-house. Building up domain knowledge takes considerable time. Moreover, this is effectively an investment in the outsourcing supplier, and not the customer's own development staff. For certain outsourced tasks that involves standardized (non-differentiating) technology, this may be an appropriate strategy, and may pay off when a company is building a long-term relationship with a supplier.

Another lesson learned by Sony Mobile is to evaluate carefully what should be outsourced. Sony Mobile has extensive experience with outsourcing, and this has led to the development of a global software outsourcing strategy. They also introduced a shared outsourcing forum for their global development centers, which had been struggling with different outsourcing projects for years. The global outsourcing strategy defined two major activities.

www.dbooks.org



The first activity was to secure a global alignment of introducing outsourcing activities and outsourcing partners. Projects, partners and all tasks, risks and issues involved in an outsourcing project should be managed systematically and equally. This way, it's possible to achieve synergies and identify best practices—figuring out what works and what doesn't. Security and access rights management are two key areas where it is very important to use common best practices, because those are critical to Sony Mobile's products.

Furthermore, Sony Mobile created a common reference process framework for analyzing, preparing and executing outsourcing projects, which defines roles and processes for supporting outsourcing projects in organization. An outsourcing business manager supports projects in the preparation and execution phases of outsourcing projects. The reference framework also includes a milestone process for approval and execution of each outsourcing projects, which helps to keep track of the stages of the various outsourcing projects within the company.



The second activity was to create a decision framework to help business units analyze and select suitable components to outsource. Using a tool support, business units can evaluate components based on a set of three key parameters.

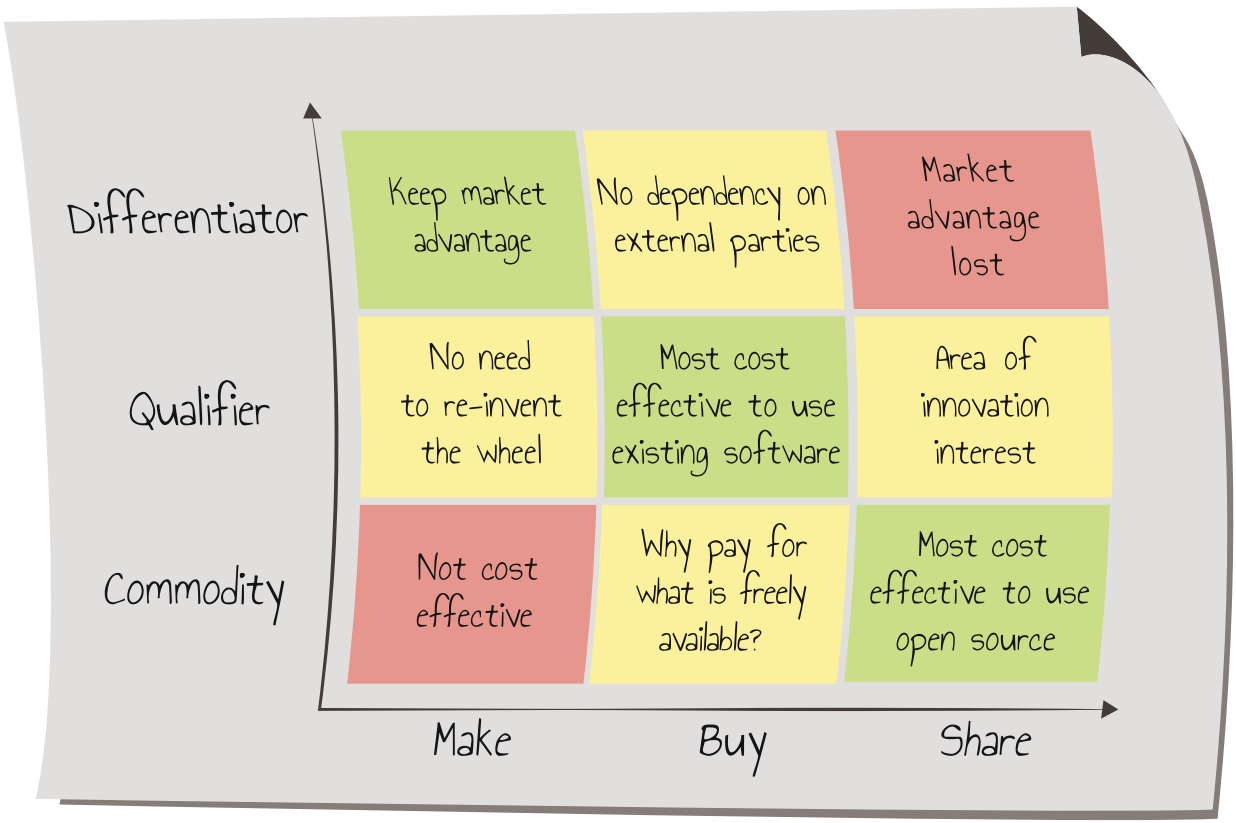
The first parameter refers to **current capabilities**, which refers to competence and amount of resources. What is the current capability for a given component? Is there a lack of competence to implement or maintain the component? Are resources wasted on components that can easily be acquired from a third party supplier?

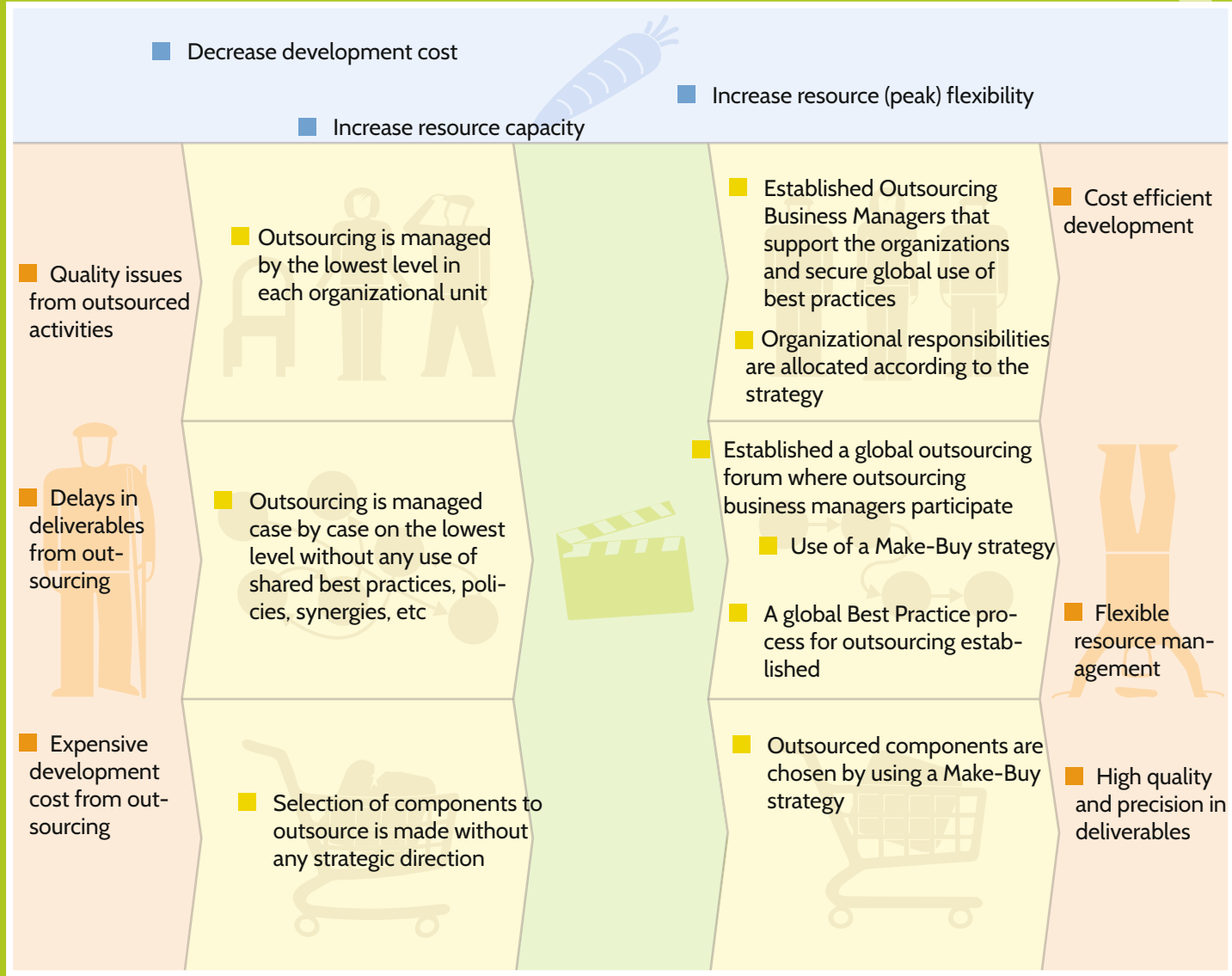
A second parameter is **dependencies**, including technical and project dependencies. Technical dependencies indicate the extent to which a component is coupled to other modules in the system. Project dependencies indicate the level of usage (or reuse) of a given component by other projects. If a component plays a key role in many systems, this means it is important to an organization as a whole, and such components should not be outsourced.

The third parameter is concerned about **long term control and competence**. This is an indicator of whether or not it is important to be able to control a given component's roadmap and future evolution. When outsourcing (or opensourcing) a component, a certain level of control is lost. Components that represent key assets (or "crown jewels) of a company, Sony Mobile have found it is best to retain development in-house.

If, on the other hand, components are 'commodity assets,' a company will get very little differentiating value from such components. In such cases, it may be a suitable candidate to outsource. Typically, excellent candidates for outsourcing are software assets that are in the maintenance phase or better still, in a dead-end state, where no or limited reuse is to be expected.

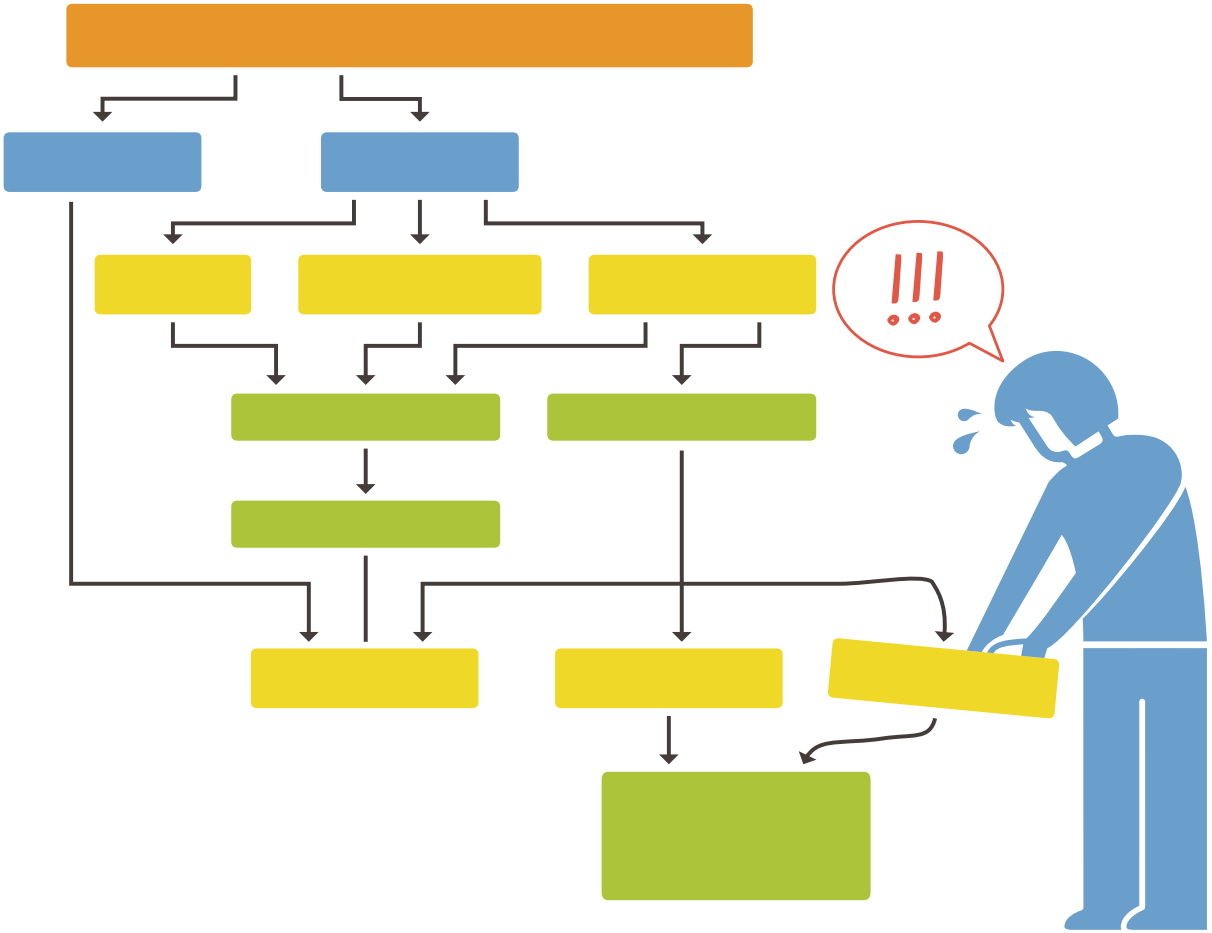
www.dbooks.org


Differentiator
Qualifier
Commodity
Keep market advantage
No dependency on external parties
Market advantage lost
No need to re-invent the wheel
Most cost effective to use existing software
Area of innovation interest
Not cost effective
Why pay for what is freely available?
Most cost effective to use open source
Make
Buy
Share



www.dbooks.org

Not so shore anymore



This is the story of a business unit in a large multinational organization that decided to outsource all development and maintenance of one of their systems. The system was a large product lifecycle management system, which was of critical importance to support the organization in its functioning. The system's architecture was typical for this type of systems, and the system itself was a configured commercial system that was tailored to the organization's needs. Furthermore, the system architecture was extensible by including custom developed modules.

In the years prior to the decision to outsource further development and maintenance, the organization had benefited from considerable economic growth. However, more recently the organization suffered from an economic downturn, which led management to establish cost saving strategies. At the same time, the organization had moved away from a traditional waterfall development process to agile approaches, but still retaining the waterfall's model focus on defining functional specifications – the resulting process was therefore a hybrid one. The development organization was still divided in maintenance and development; solution managers were responsible for the contact with the organization by using the system and develop functional specifications with architects and developers.

In order to cut costs, the organization decided to outsource all development activities. In addition to the cost-saving driver, another driver was to become more flexible in spending resources on new functionality. The company chose an existing outsourcing supplier that was already used for other large systems within the organization. The organization's requirements on cost savings resulted in a decision of the outsourcing partner to offshore the entire development to another organization in India.

The solution manager and the architecture knowledge were retained in the original organization, while development was moved out. The outsourced part consisted of about ten developers and ten testers. It was clear that a process based on specifications was needed, which meant that the previous agile transformation was abandoned, and a waterfall model was adopted instead. An implication of this was that the on-site roles became less interesting from a technological point of view, which resulted in the architects leaving the project. This in turn led to a reduction of skilled and experienced staff that was available which was very important for the requirements and design phase.

www.dbooks.org

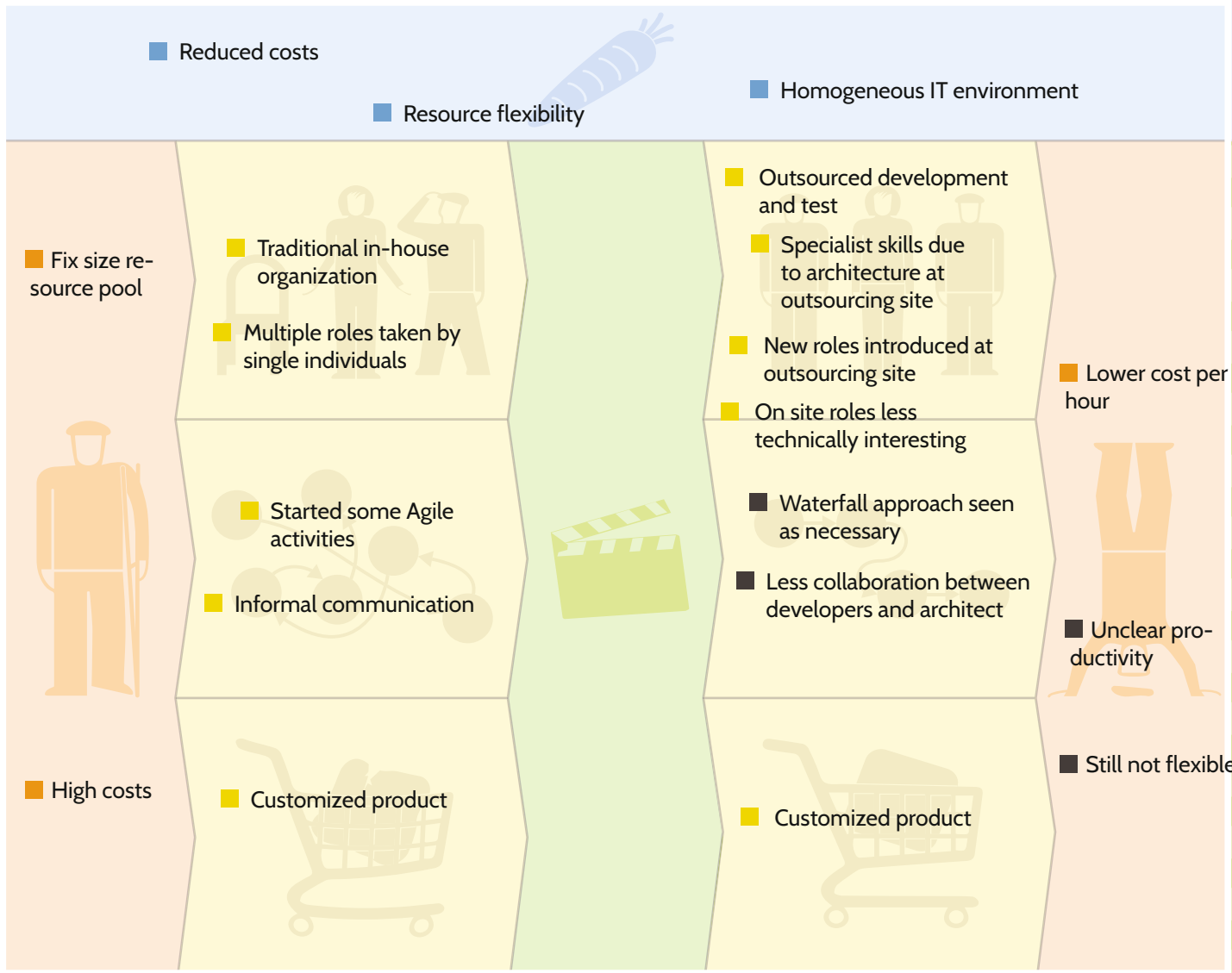
At the outsourcing organization there were developers as in the original organization, but also team leaders responsible for leading the work and keeping the contact with the original organization. Turn-around times for development became longer and it became much more difficult than anticipated to find more staff at the outsourcing organization when more development needed to be carried out. All the customizations became roadblocks. These forced the developers to undergo a large amount of training before they could be productive. But, despite all the training, they also misunderstood the specifications, resulting in a large number of problems at the first major release.

The initial phase was characterized by long lead times, a large number of misunderstandings and low flexibility of ramping up/down the development force when this was needed. They focused too much on the formal process instead of having a common view of the development and the system was simply too customized.

After the initial outsourcing phase, efforts were made to improve the understanding of the development from both sides of the organization. Representatives from the outsourcing organization came to visit the original organization. The original organization started also to visit the outsourcing organization more frequently. This resulted in both better understanding of the development and in a more personal commitment to the system and the original organization at the outsourcing organization. However, the architecture was still too customized.

A number of recommendations can been made. Creating a common social group for developers early in the change process would probably have worked better than the starting off with a formal process based on specifications and hand-overs. It would probably also have worked better if the system was less customized, which would have made it possible to find the right competence already from the beginning at the outsourcing organization. The developers at the outsourcing site could probably have been involved earlier to reduce mistakes and to gain a better understanding about the system and why it was needed.

The organization decided later to take home the development and conduct it at another business unit inside the company. The driver for this was also this time to save costs, but which this time was possible as the business unit that will do the development had about the same cost of developers as the outsourcing organization. With the same cost structure, but with a development organization closer to the users, we simply get a more efficient development process.


Reduced costs
Resource flexibility
Homogeneous IT environment
Fix size re-source pool
Traditional in-house organization
Multiple roles taken by single individuals
Started some Agile activities
Informal communication
High costs
Customized product
Outsourced development and test
Specialist skills due to architecture at outsourcing site
New roles introduced at outsourcing site
On site roles less technically interesting
Waterfall approach seen as necessary
Less collaboration between developers and architect
Customized product
Lower cost per hour
Unclear pro-ductivity
Still not flexible

Play it again, Sam, backwards



PlayNow was Sony Ericsson's download service for media such as music, games, ringtones, wallpapers and themes. They used to have the bigger part of the PlayNow team outsourced. The development was taken care of by the outsourcing partner and Sony Ericsson took care of the project and product management internally. The outsourcing partner actually desired to have the project and product managers at their site to drive the software developers more efficiently, but this never happened. So the set up was very top heavy. Communication could only go between a point-of-contact at each company, causing a lot of time lost. Every three months the outsourcing partner delivered a version of the software delivery. This led to a very slow feedback loop. It was difficult to know if what had been delivered also was what had been developed.

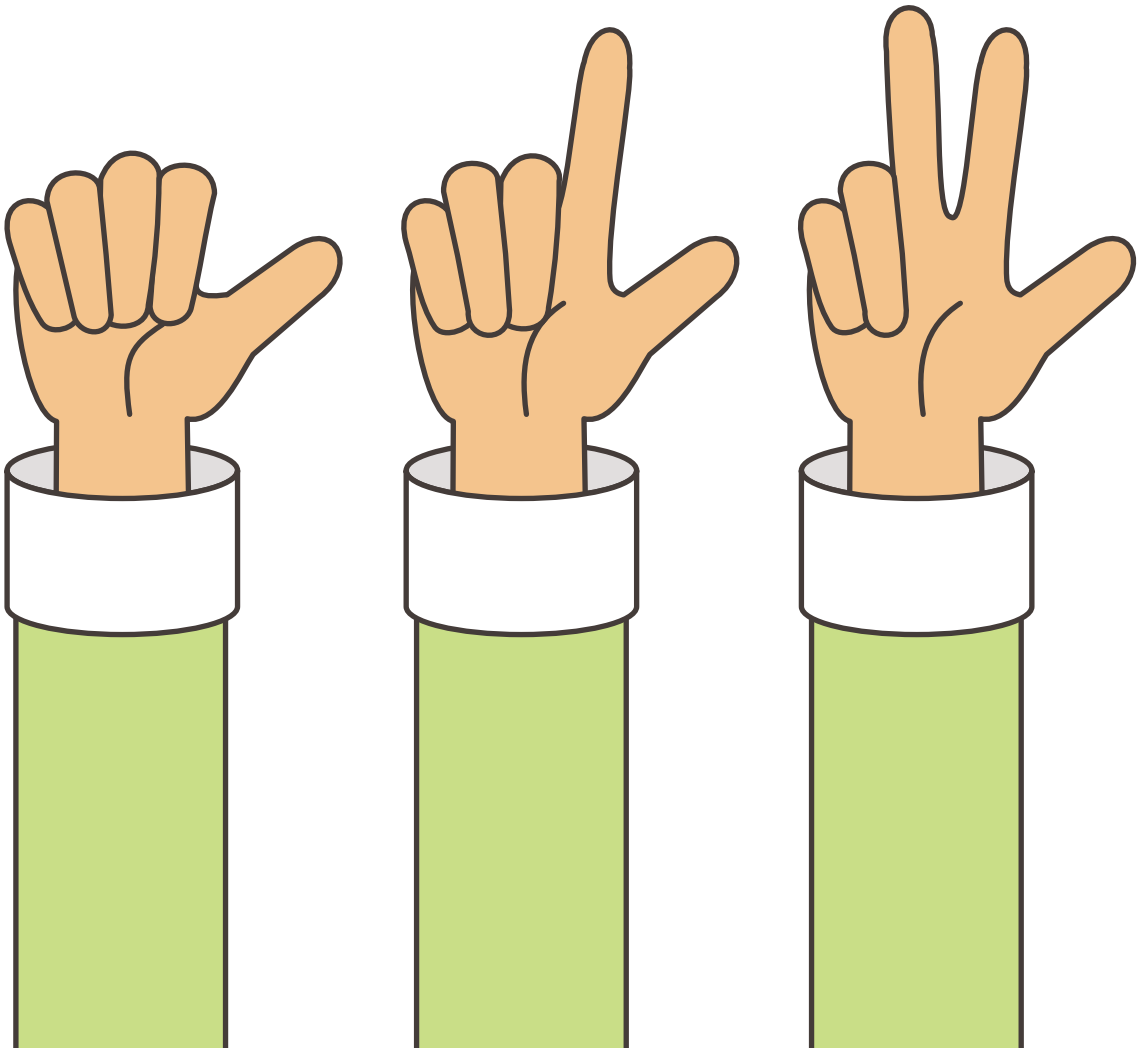
Due to cost saving directives, management decided to bring back the software, to develop it in-house. This proved to be a more efficient way to work. The cost saving was approximately 75% even though cost per head-count was increased. This was mainly thanks to reduction of overhead and that they started to work in an agile way with weekly deliveries.

What we can see from the Sony Ericsson case is that it is not always cheaper to outsource. Overhead, communication and innovation were factors that certainly added extra cost to their outsourcing activity. This is not an isolated case, several other companies suffers from the very same problems. A global trend is that the outsourcing market is shrinking. The largest outsourcing deals in the world are far less valuable today than they were ten years ago, according to IDC in the Wall Street Journal.

To decrease costs is one of the most common reasons to outsource, but outsourcing is not always the least costly solution. As in this case, the overhead cost of outsourcing grows that much that it is a lot cheaper to bring home the software development. By having the software development in-house it's easier to keep the project in control and to know what is developed and why. Those aspects are much harder to manage when all development takes place outside of the company walls. Another reason that causes added costs is the growing overhead in the outsourcing organization. Usually only software development costs are included in the cost calculations. But, the outsourcing partner also needs to have project managers, architects, system designers and line managers.

www.dbooks.org

First things first



"What to do"
is about finding the requirements from the customers and to communicate them to the software developers.

"How, when and where to do it"
is about the software development methodology, how we take on roles and responsibilities and split the organization into a sensible structure. It's about staying in control regarding your source code. Which revision should we work on? Which part of the software have certain part of the functionality?

The first things are about the very basic needs. We need for instance to know what to do and how, when and where to do it. We also need to check the quality of it.

"Check the quality of it"
is simply to follow up on the planning and the coding (by defining and conducting tests).

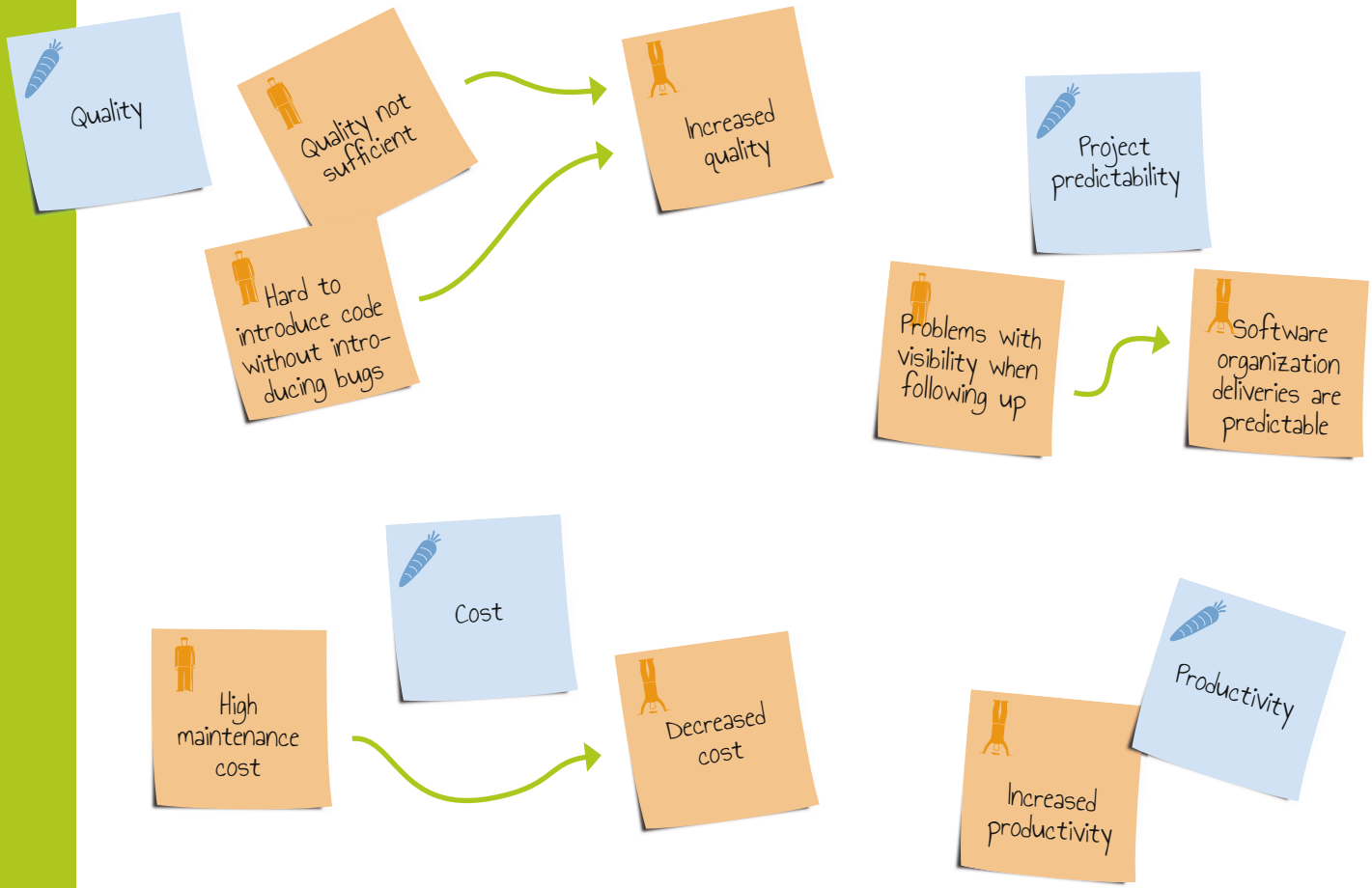


A good read

Basic principles and concepts for achieving quality, Emanuel R. Baker, Mattew J. Fisher, 2007

www.dbooks.org

This might not come as a surprise, but in case our organization has been growing from just about nothing to employ some twenty developers or more, it's likely we have problems with insufficient quality and increasing maintenance costs. We have probably problems with visibility when following up on projects and it's likely difficult to make bug corrections without causing new bugs.



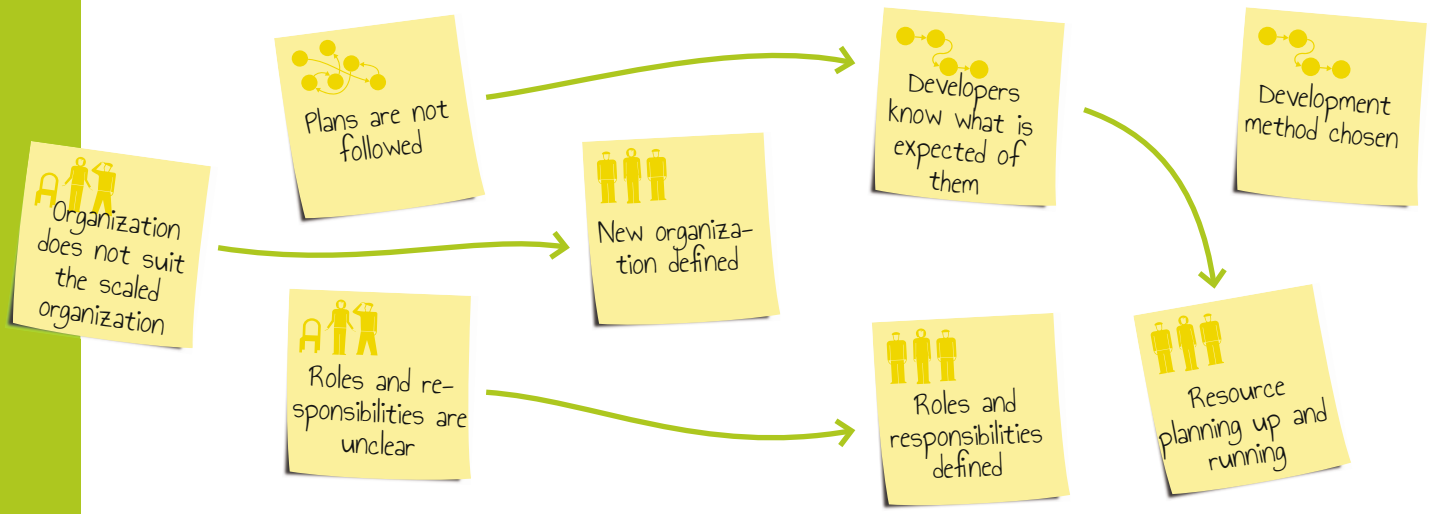
One of the reasons why we have these problems is probably that our organization has outgrown our current way of working. Redefining the way of working to match the current size of our organization will in general lead to increased quality, productivity and efficiency of the business. It will in particular make the projects more predictable.

It's hardly a surprise that growing organizations get problems. To do a good job, basically developers need to know "what to do" and "how, when and where to do it". As easy as one, two three, one might think. As a matter of fact, it gets more and more difficult to spread information as the organization grows. What isn't a problem to communicate between very few developers, simply is more complicated in a larger team. It isn't rocket science. We recognize this from all sorts of contexts, not just business. Yet most software projects fail in their communication. Only a software organization in possession of these basic capabilities can be scaled to meet the demands of today; if not with ease, at least without the chaos it would cause to not have them.

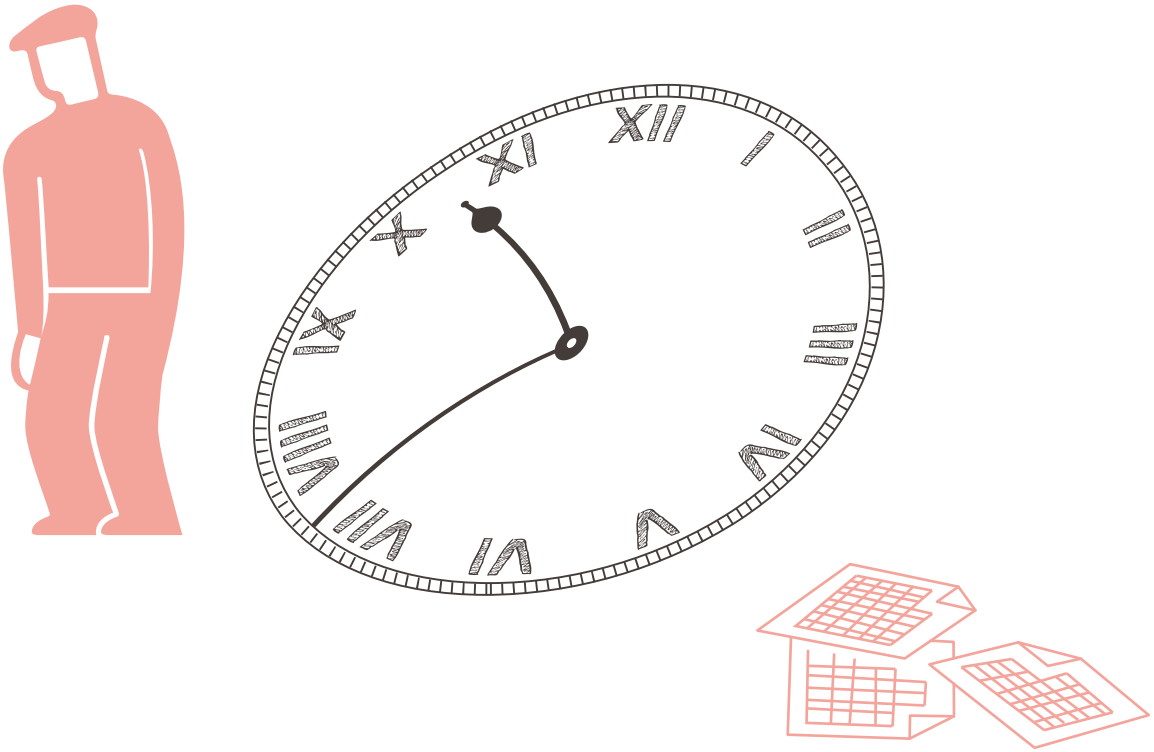


To state the obvious, our engineers need to know what to do and how to test what they have done. The requirements need to be communicated in a way that they understand. While there are many ways to manage requirements and every way comes with its pros and cons, most important is that we do it.

Growing the software labor effort from one or two developers to more than 15 to 30 developers puts great demands on both the software process and the software architecture. While two developers could handle requirements, configuration management, quality assurance, project planning and follow up in an informal way, the larger team simply runs into serious communication problems.



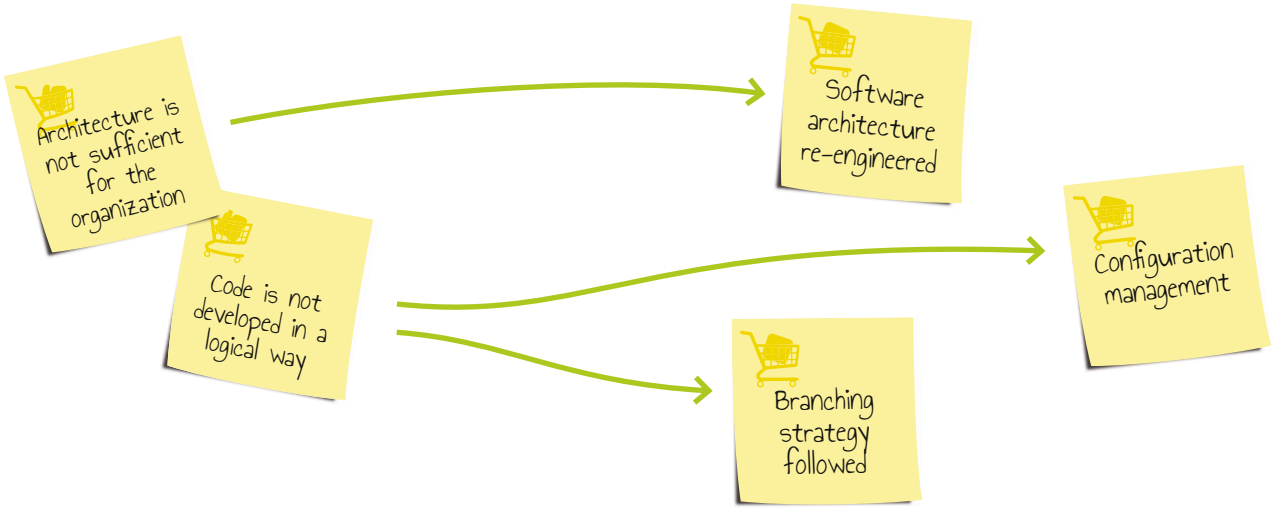
All engineers need to know their roles and responsibilities. The organizational structure must facilitate effective communication in all directions, not just vertically. The optimal structure closely follows the ways people work, weather they work in projects or in a product line. Agile development methodology, which nowadays is the de-facto way of working, promotes for instance self-organized teams that in its most extreme implementation can be seen as companies in the company. It's safe to say that we should avoid old school hierarchies that merely organize people on what they do.



About those ever-late projects of ours, they need to be dealt with. It's not that our current project managers aren't doing their job; it's just that they have to change direction every now and then. The ways to plan have to be adapted to the reality, where plans are revised continuously. It's wise, though, to not overdo the planning and instead try to capture the few next weeks in detail. In Agile development methodologies, already mentioned, all planning is made in two-week chunks. Trying to grasp a much longer period of time into a plan has simply shown to be doomed to fail.

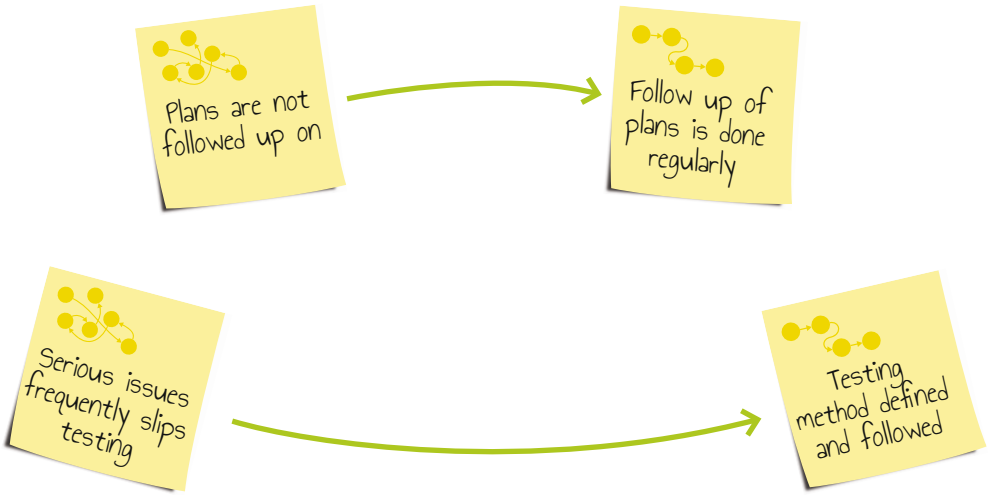
In what state is our software architecture and how do we manage it? The software has to serve its purpose, to fulfill the function for which it was created. It must also be firm in terms of structural integrity and durability. If optimally designed we should be able to:

- Make changes in one part of the system without negatively affecting other parts
- Distribute work in the system between different departments
- Reuse software components from one part of the software system in another

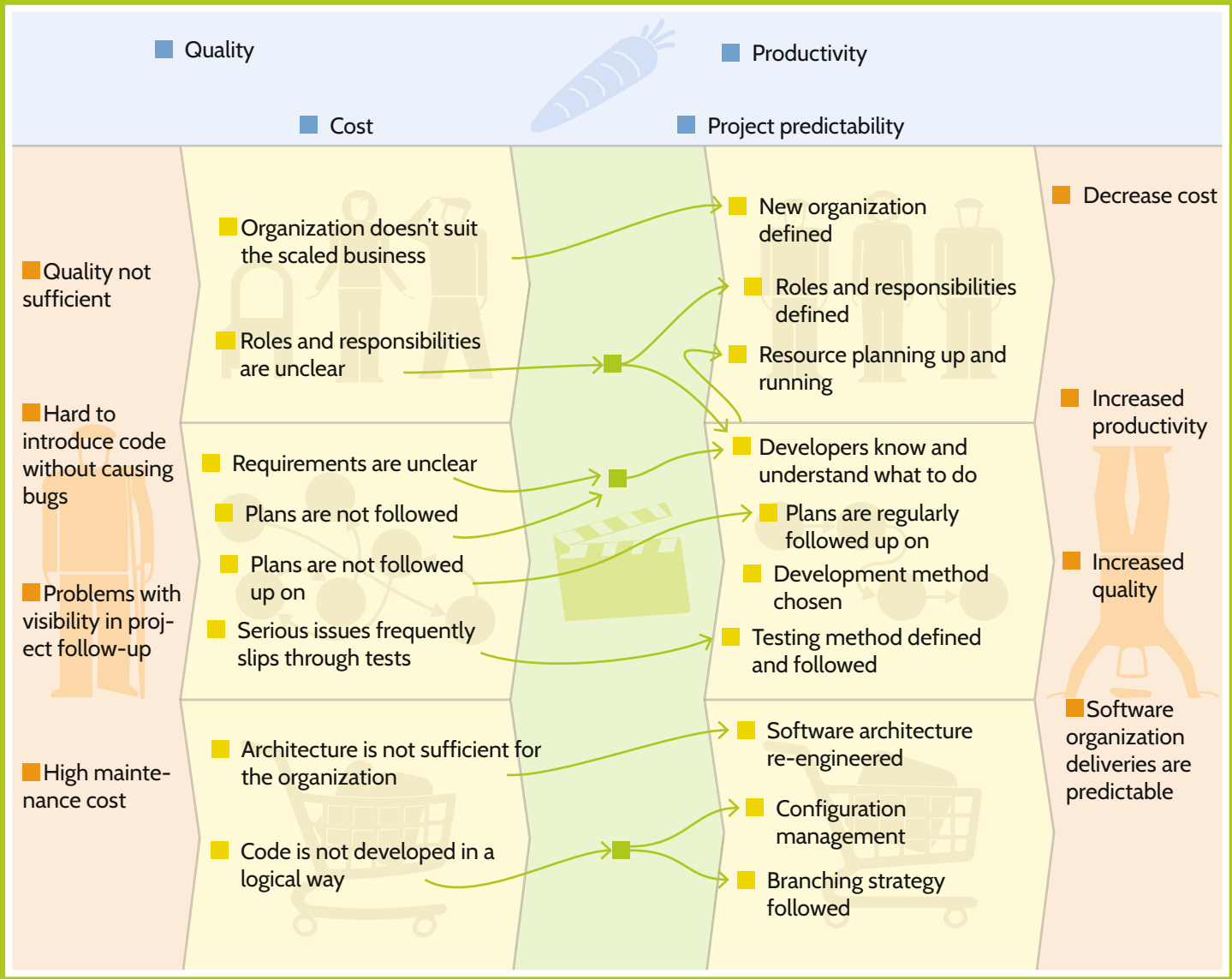


It's key to design the system as a set of clear-cut parts embodying well-defined boundaries. Software architecture is, simply put, how these parts are structured and how they relate and communicate with one another. The architecture starts with its documentation, a blueprint that governs how to design parts in order to facilitate development of the software system. And, as with ways of working has the organization to tightly follow the architecture.

Finally, we have the testing. Is this activity considered the necessary evil that continuously gets down prioritized when the project slips in time? If so, we should really be cleverer. To monitor and evaluate how the organization is doing quality-wise pays of as soon as the heat is turned on and the business gets into full production mode. When everyone involved runs as fast as they can, occasionally even making short cuts to get in time, there's simply no room for thinking about what can be improved. The test activity is really our only mean to identify bottlenecks and to optimize processes and tools.



Optimally, to ensure we're going in the right direction, both the plan and the software ought to be followed up on. This shouldn't be done at the end of the project but frequently, tightly coupled with the iterations in which development takes place. We would want to keep the feedback loop as short as possible. All Agile development methods have this built-in to the method. At the end of the iteration, an automatic test script would test the software and a retrospect would evaluate if we work in a good way or if a change of direction is needed.


Quality
Productivity
Cost
Project predictability
Quality not sufficient
Hard to introduce code without causing bugs
Problems with visibility in project follow-up
High maintenance cost
Organization doesn't suit the scaled business
Roles and responsibilities are unclear
Requirements are unclear
Plans are not followed
Plans are not followed up on
Serious issues frequently slips through tests
Architecture is not sufficient for the organization
Code is not developed in a logical way
New organization defined
Roles and responsibilities defined
Resource planning up and running
Developers know and understand what to do
Plans are regularly followed up on
Development method chosen
Testing method defined and followed
Software architecture re-engineered
Configuration management
Branching strategy followed
Decrease cost
Increased productivity
Increased quality
Software organization deliveries are predictable

Get inspired

Being one of the engineering disciplines mostly written about, you shouldn't have to go far to find in-depth success stories and lessons learned from diverse efforts in straitening up software organizations and architectures. The following pages summarize the real-life case studies that were conducted to define this scenario.



Robotic growing pains

Read about Husqvarna's experience when they added Internet of Things to some of their lawn and garden products.



Softhouse reflects on architecture changes

Learn about the effects of architectural changes in Android development.



From mobile to platform

The platform concept builds on components that easily can be changed or configured. This enables new products and offerings to be created quickly, without having to redo a lot.

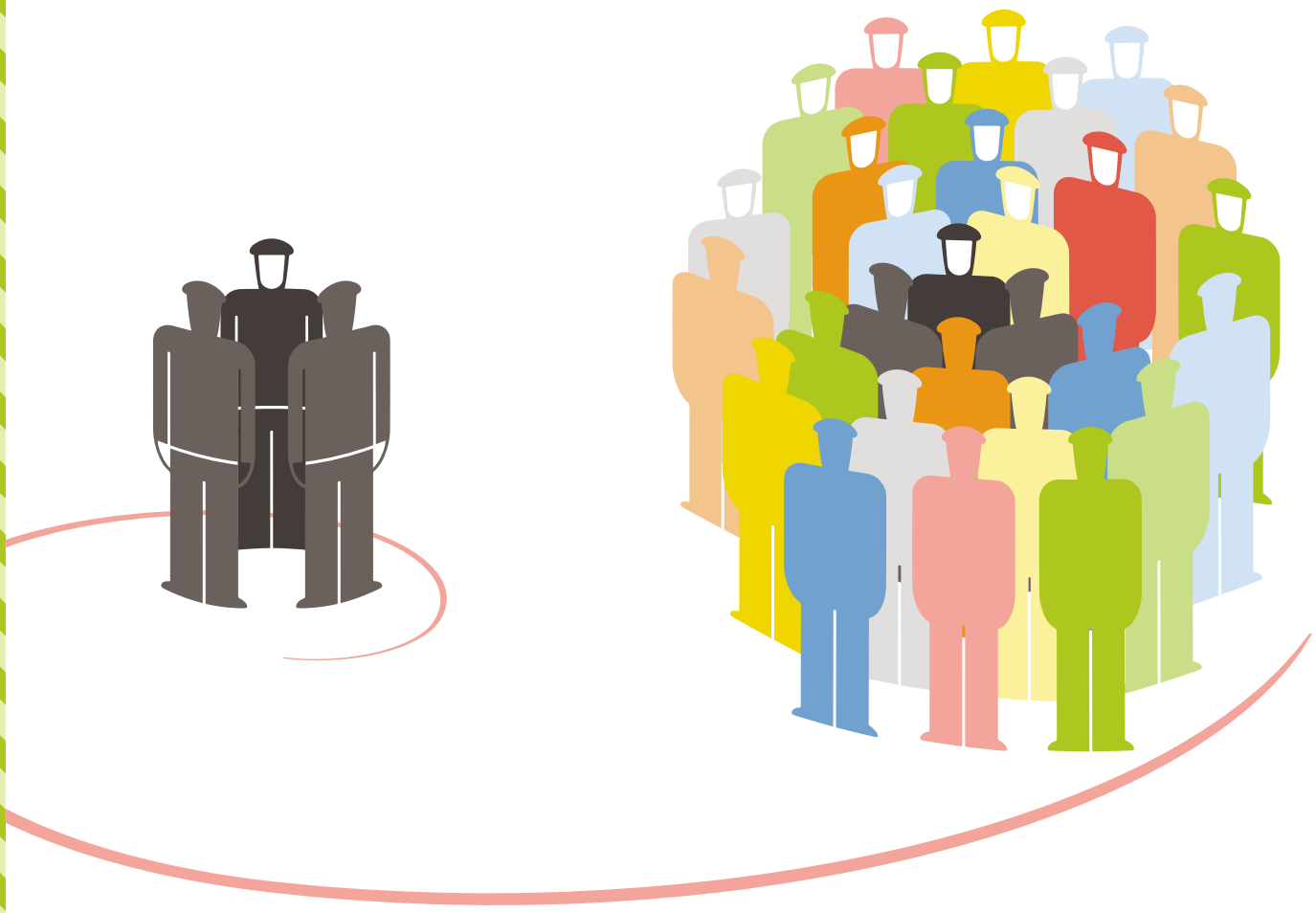


One more thing

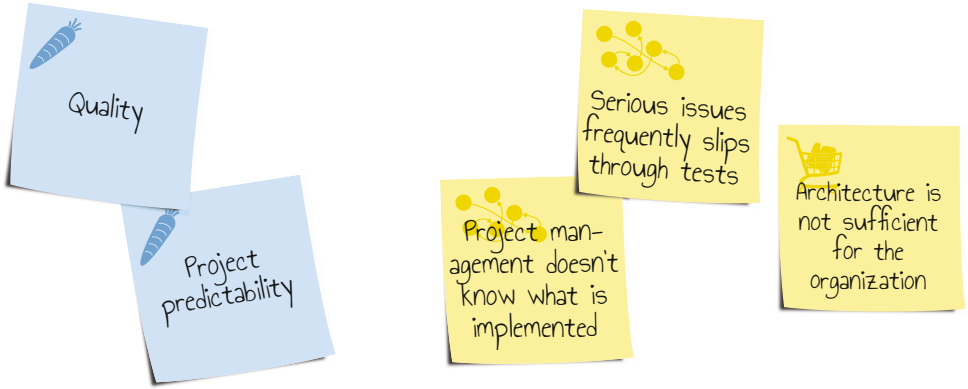
When scaling through *First things first* a development method has to be chosen. As we have hinted throughout this scenario, it is strongly recommended to work in an agile way. Find canvases for Agile development in the chapters **Pump up the volume**, **Deliver 24/7** and **Agile and disciplined**.

www.dbooks.org

Robotic growing pains



Husqvarna Robotics had grown from a small team of 3 software engineers to over 30 software engineers in a very short time. They understood they needed to improve their quality and project predictability because of the problems they had with their software development.



Robotics had started to get problems with late deliveries, bug corrections very often led to new more serious problems, and new bugs reports from the market disturbed the software team in their work to develop new functionality, this led to delayed software projects and releases. Project managers were not confident that the software team could deliver according to the plan. Architecture improvements were pushed in the future, due to lack of time. The testers did not have time to test everything in a release, which led to even more bugs being reported from the market.

Project management did not know which requirements being implemented at the moment. There was little visibility of the progress in the software team. All software was delivered late to the main branch. At that time a major part of functionality did not work, so what was working or not wouldn't be discovered until very close to the release of the product.

Product management did not have confidence in that the software organization was delivering new functionality efficiently.

www.dbooks.org

During fall 2014 a series of general seminars in software engineering was conducted in the entire R&D organization. Robotics understood it would be good to get external help to do an analysis of the current situation. A kick-off of the improvement project at Robotics was held in September 2014. A series of interviews was held and a report of the situation and improvement proposals was presented in January 2015.

Suggestions of improvements were made for the areas such as requirement handling, project planning, project tracking, software quality assurance and configuration management. It was also said that the software architecture needed to be restructured.

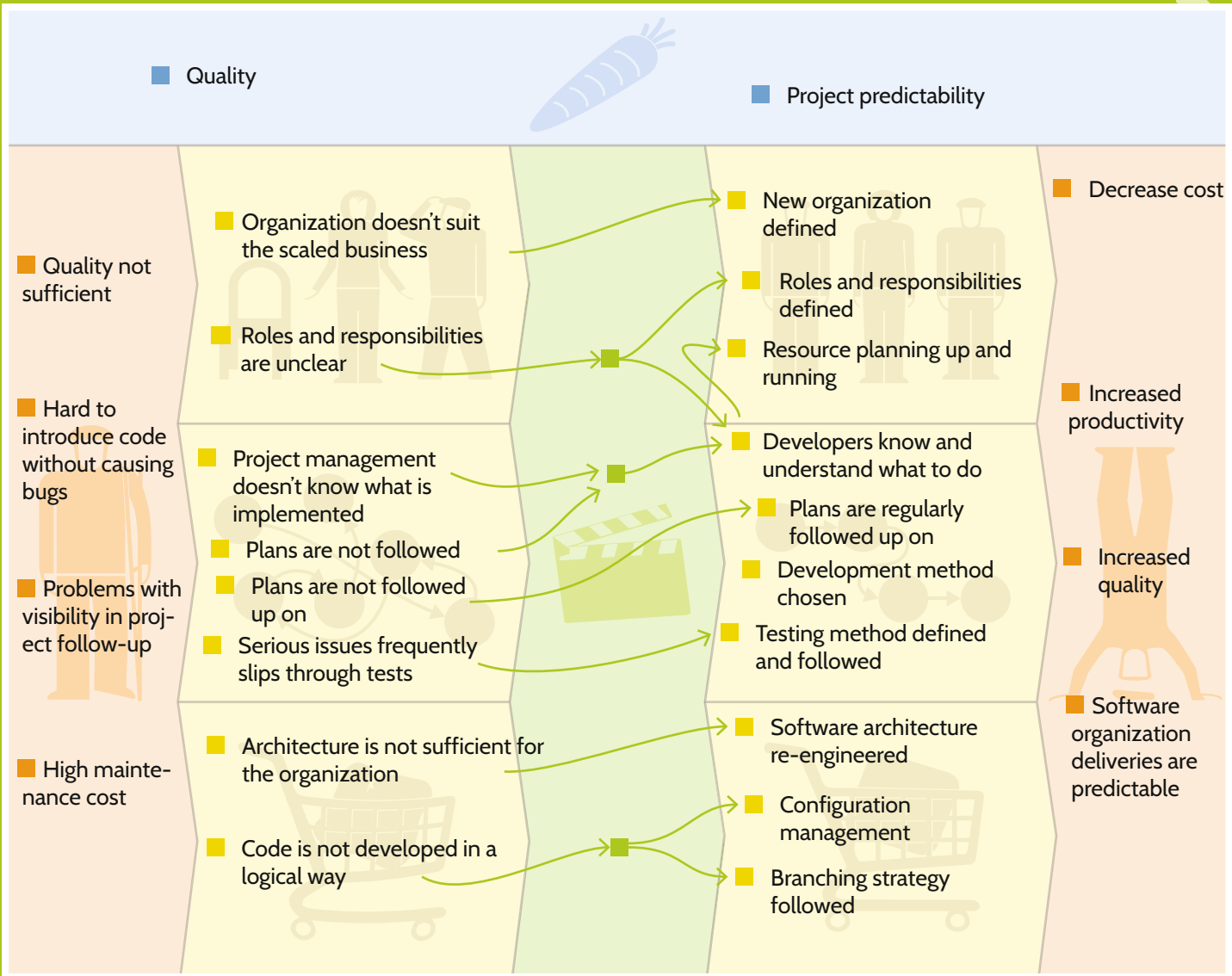
A new meeting was held to agree on which changes to prioritize. The software organization, project management and product management were all involved in this decision.

The most important suggestions of improvements from the audit were prioritized to be implemented during the first half of 2015.

In April 2016 the software organization had a completely new situation. Instead of the negative atmosphere that characterized 2014, a more positive attitude characterized the organization. The software team was positive to the new ways of working and they believed they were working in a good way.

By the end of 2016 the initial team of three had left the company. They did not find their place in the larger organization. Even though this was a large competence loss, the organization was now better prepared for situations like this. The new ways of working and the new documented architecture made the organization less dependent on a few very competent champions.

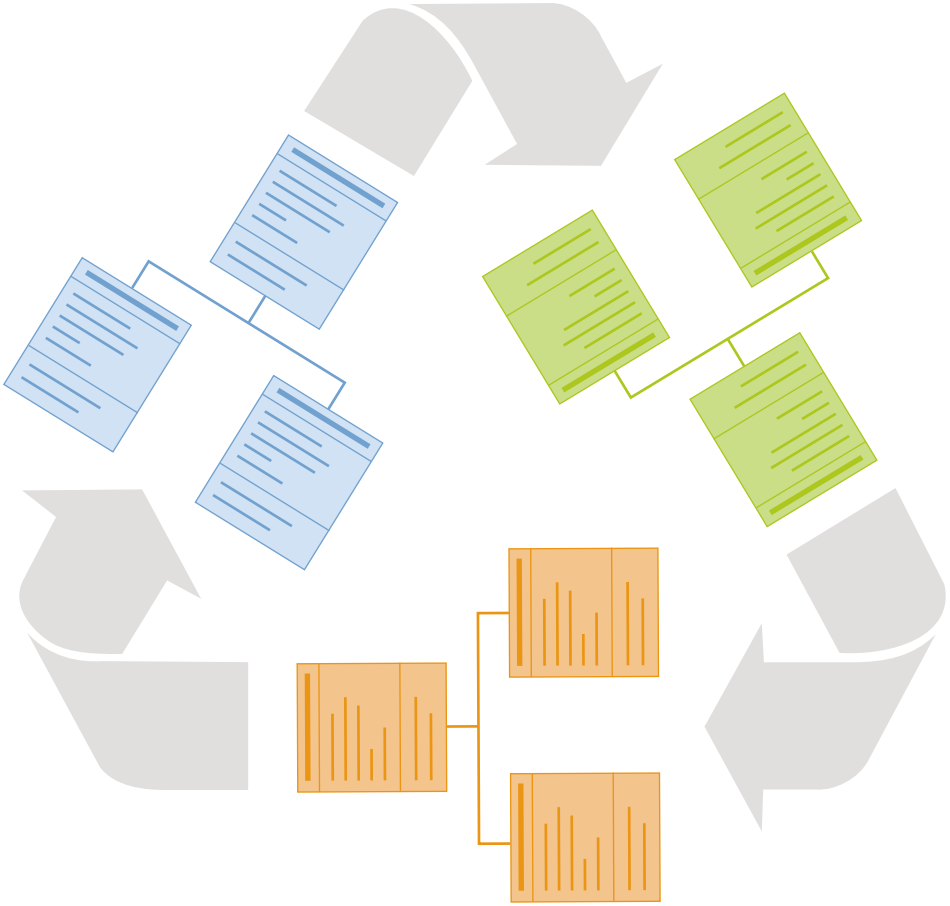
Husqvarna has in this project gone through a very common growth problem. The exact same problems were found in several different Ericsson departments in the early nineties and have been found in several other companies since then.



www.dbooks.org



Softhouse reflects on architecture changes



A typical situation in software development is that a product is developed as a prototype or as a first version but in a rather small scale, and then the product and the number of included functions grow. One problem with this is that the software architecture might not be suited for what it is used for, thus resulting in added functionality that causes unpredictable software faults. It gets necessary to improve the architecture through refactoring. This happened to Softhouse.

There were three main reasons for Softhouse to make a change in the software architecture:

- The amount of code and functionality a new team member needed to understand was too large.
- There was a negative trend of quality issues like old bugs being re-introduced and too many errors found late in testing.
- The software system would be significantly extended in the near future and be used in new ways.



Starting as a small prototype, they created a client system to provide their customers with data from their internal information systems. As the usage of the client system grew, the product itself and the number of functions increased rapidly. A problem was for instance that any changes in the product affected many parts of it, which resulted in unpredicted faults.

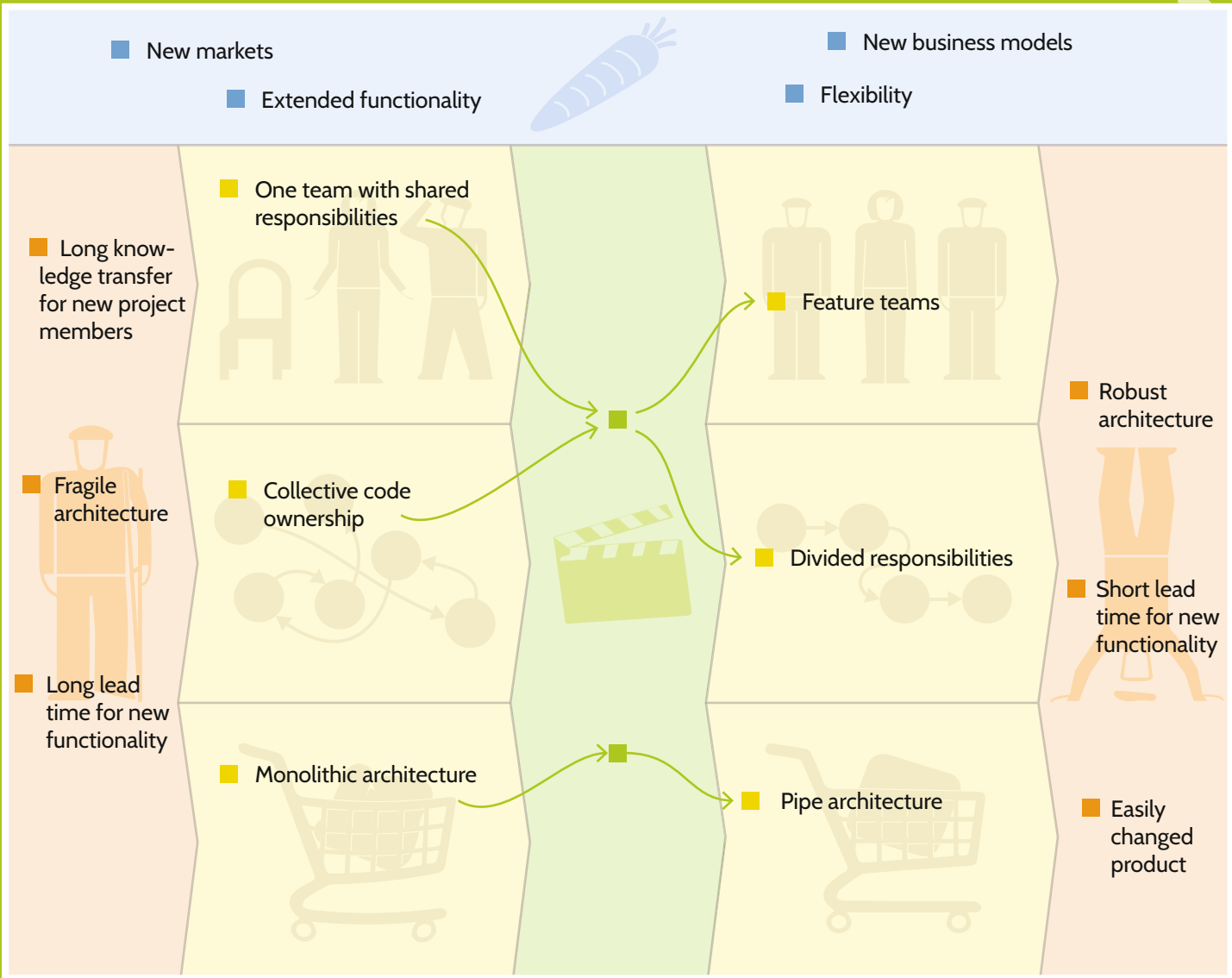
The system was built with focus on reuse, i.e. when new functionality was added, existing classes were reused as much as possible. This focus lead to an architecture with many dependencies resulting in a monolithic system. This was identified as the reason behind the many quality issues.

www.dbooks.org

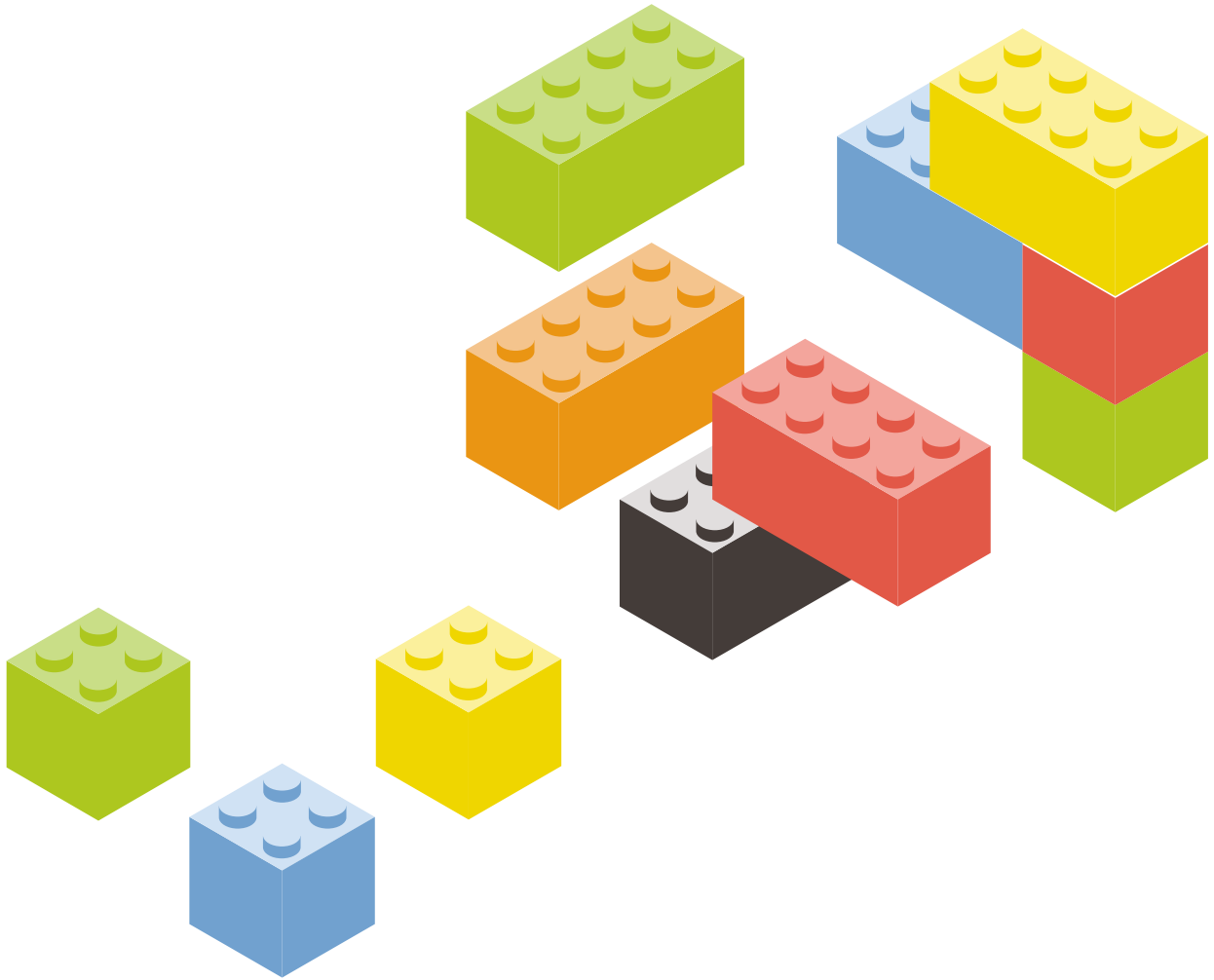
To make the design less fragile, the architecture was divided into modules. Each module implemented one specific function provided to the user. Functional decoupling allowed the developers to make corrections or updates of existing functions more efficiently and with higher quality. It also allowed for parallel updates of different functions at the same time. It also allowed for introduction of new functionality independent of the existing ones.

The architecture guidelines were changed to stress on the use of independent modules and how to manage them individually. Drawbacks of architectures like this are less reuse and more double maintenance of similar code in different modules. However, changes can mostly be limited to one module and therefore fulfill the maintainability requirements.

The project followed an agile approach similar to Scrum with collective code ownership where the developers assign the tasks to themselves. The new process allows developers to avoid change requests in modules they haven't knowledge in. From an organizational perspective, the new architecture makes it easier to scale. New developers that join the project can start work on one function in one module and learn the system function-by-function. The new software architecture is now used in full effect.


New markets
Extended functionality
New business models
Flexibility
One team with shared responsibilities
Long know-ledge transfer for new project members
Feature teams
Robust architecture
Collective code ownership
Fragile architecture
Divided responsibilities
Short lead time for new functionality
Long lead time for new functionality
Monolithic architecture
Pipe architecture
Easily changed product

From mobile to platform



To move towards a platform development strategy, where the same software is used in many products, is quite natural and a common strategy in many companies. After the first product has been successfully released, the company now looks for ways to grow the business and to reuse the investments already being made.

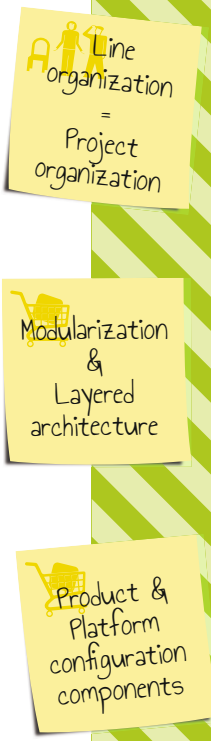
The platform concept builds on modularized, stable and reusable components that easily can be changed or configured. This enables new products and offerings to be created quickly, without having to redo a lot. However, the strategy and its implementation is always an act of balance between reuse and product focus.

When Sony Ericsson started to take off after the success with their first mobile phones, all their activities took place in product projects. There were two similar product development organizations with redundant development competence as a way to develop more than one product at a time. After the first products had been released there was a need to increase the business and offering even more. It was of course not possible to scale up the development capacity linear to the number of products in the portfolio. So a platform concept was introduced.

The old software architecture was heavily impacted by the platform concept. Modularizing and layering of the architecture together with a configuration and customization framework were key concepts and patterns in order to create the platform concept. It was essential to identify common functionality and things that needed to be customized, in creating the configuration and customization framework.

This made it possible to maintain and reuse the majority of the software and functionality and thus only work with configurations and some product and customer specific development. This also led to that the number of product variants grew dramatically.

This in turn had an impact on both the processes and the organization. The software configuration department grew and became the heart and engine of the development. A special software release management organization and process were established to cope with the large number of variants. How to test and verify in a cost efficient way be-



www.dbooks.org

came the billion-dollar question, as they couldn't multiply the test activities in a linear way. They had to find ways to also reuse test and verification in a smart way.

Sony Ericsson was successful in this journey since they were able to release more and more products without having to increase the work force in a corresponding way. The time-to-market and cost per product reduced significantly.

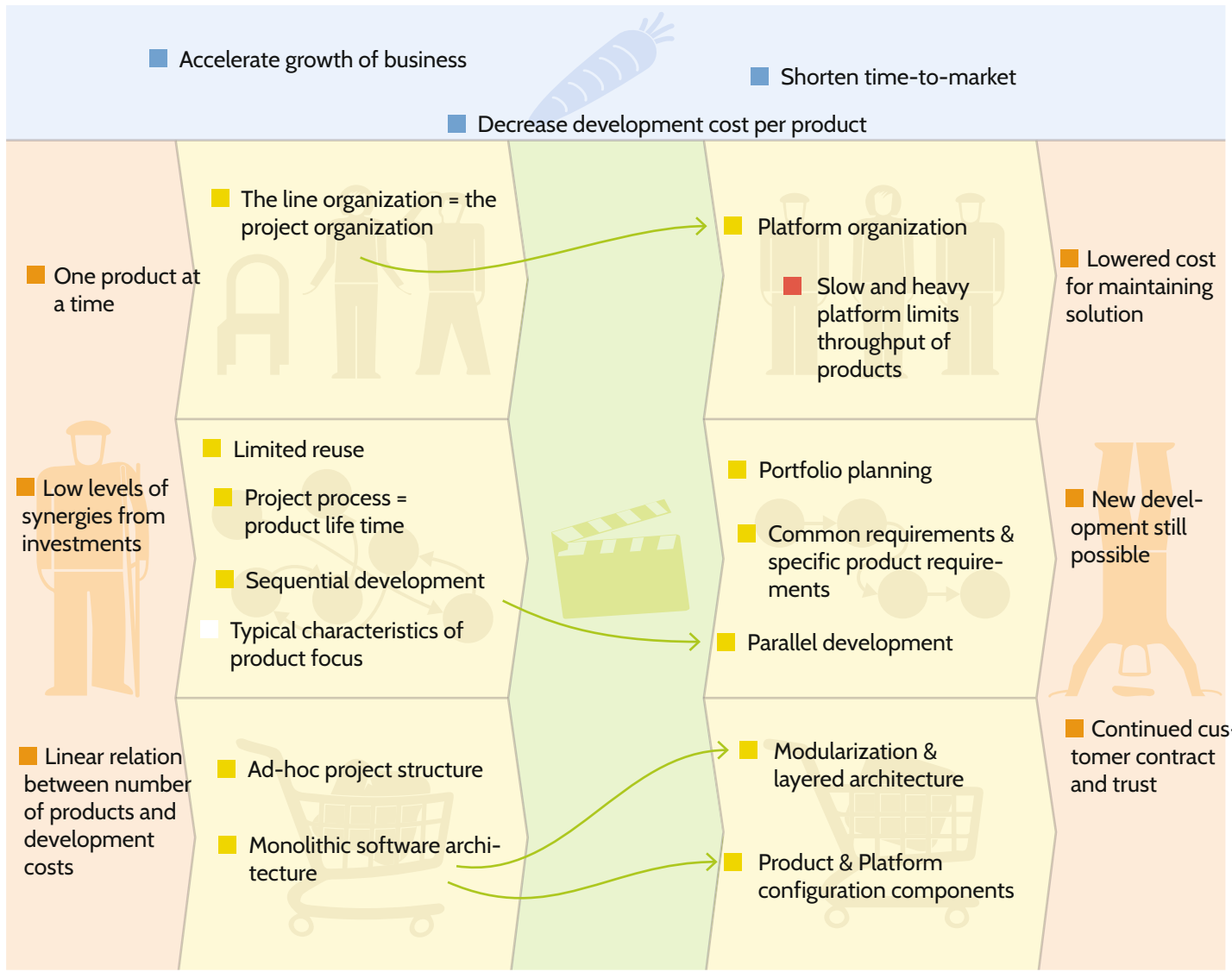
– So did they continue to improve and become better and better?

The answer is no.

As the platform concept grew stronger and stronger the focus on the product itself and its offering became weaker. The platform projects and its organization grew bigger and bigger and tried to include more and more products in its releases. Together with a waterfall approach this led to that the platform projects became slow and inflexible giants. These tried to please all products with all their market and customer needs. This also made the projects lengthy as not all products were released at the same time. As a way to cope with all the products and requirements, the platform projects started to claim that all requirements had to be set at least two and a half to three years prior to the product releases. This was not possible in the mobile industry during mid-2000, as tons of new features and concepts were released every year.

The organization and process became impossible to maneuver and the product offerings became late and were not competitive enough on the very tough mobile phone market.

The lesson learned is that a platform concept that is well prepared and balanced with a continuous product and customer focus can lower time to market and reduce development cost. But it's an art of balance, to find the optimum level of abstraction, to what extent it is reasonable to reuse system components.



www.dbooks.org

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Your

© The Author(s) 2017
B. Fitzgerald et al., *Scaling a Software Business*,
DOI 10.1007/978-3-319-53116-8_3

journey

How you get there

www.dbooks.org

Good old post-its





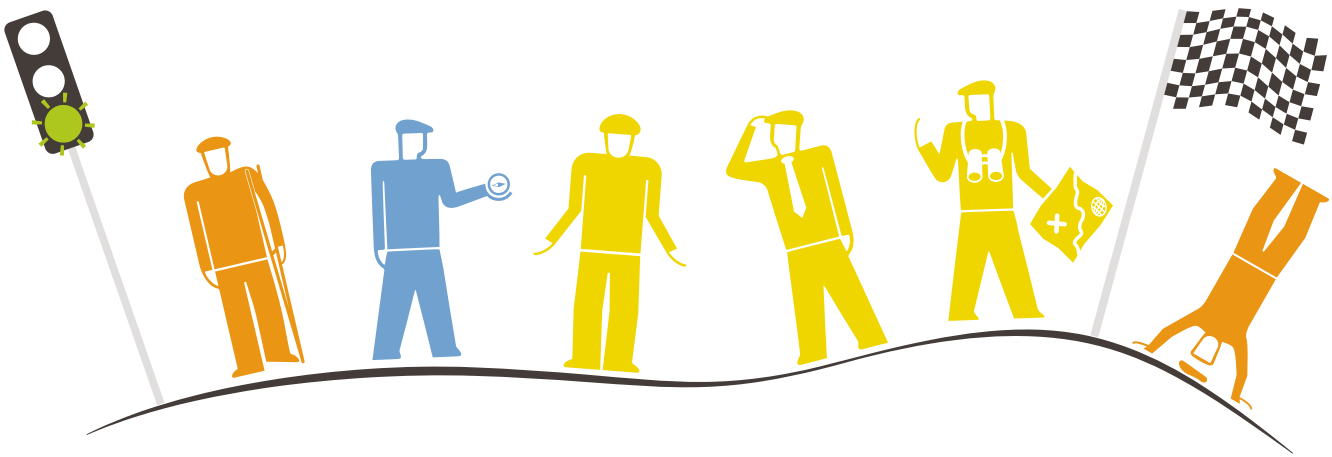
Please come in. Welcome to our home turf. It's time to roll up your sleeves and fill the whiteboard. You'll find the Post-It notes on the table. We've already put up blue ones. As usual these represent your input, what we'd like to achieve. Now, go ahead and fill the board with orange and yellow ones. Describe what changes you think it would take. Wearing your glasses, consider the impact on our organization, on our ways of working and on our offering. Needless to say, before you jot down your note, browse through the shelf with notes from previous experiences.

www.dbooks.org

Defining a transformation journey is an important step in the transformation process. In this chapter we'll show how to setup a workshop to identify the steps that an organization can take to embark on a software scaling transformation. The proposed solution from the workshop will be a concrete list of transitions describing changes in the three key domains that the Scaling Management Framework (SMF) defines: product, process, and organization. We'll use a canvas throughout the workshop to support this process.

The workshop in short:

1. Define the drivers – They should be derived from the company business strategy.
2. Decide on inabilities and desired abilities that will drive the change.
3. Identify the current domain characteristics that cause the current inabilities.
4. Use the compass to find a solution and read relevant scenarios with similar drivers.
5. Prioritize the transitions and start to implement.



Setting up the workshop

Defining a transformation process isn't just simply following a set of steps – it requires thoughtful participation and creativity. However, using the canvas to structure the ideas and drive the process will make this more efficient and more fun. It also helps in communicating the results to a wider audience, varying from other managers to developers.

Running a successful workshop depends heavily on active participation and brainstorming by all participants. It's important to get everybody engaged to encourage innovative and creative contributions. This can be achieved by actively coaching this process and asking open questions, but beware of critical comments and feedback that might discourage participants. Later on in the process, the pros and cons of different strategies are evaluated before decisions are made. We won't go into different theories and techniques on how to optimize the workshop phases and group dynamics. Instead, we'll focus on the purpose and content of each phase, and how we use the SMF and the canvas to support the process.

To get started, you'll need to have access to all decision makers in the organization. Remember to cover input from both management who understand the drivers, and from specialists from all three SMF domains: product, process, and organization. As illustrated in the introduction, it's possible to divide the work into two workshops without having all persons on site at the same time. However, when possible it is always best to gather all roles and skills in the same room at the same time, and let them work iteratively together.

Some practicalities before we start the first workshop:

Make sure that the room is equipped with a whiteboard and a projector that can display the canvas on the whiteboard. Being able to put up post-it notes and connecting them with lines and arrows is important.

Get post-it notes in different colors, preferably in five colors (blue, orange, yellow, green, and red). We use colors mostly because we find it easier to put them into context when discussing them together, to get a good overview of the situation. Actually, for most post-its, its place in the canvas tells what type it is. The color is mostly redundant, but the visual distinction can help. There is one place where it isn't redundant, so you will need at least two colors to distinguish the meaning.

www.dbooks.org



Let's start! Why are we here?

Put the SMF canvas on the whiteboard. Present the goal and the expectations of the workshop and explain how the workshop will be performed.

The goal is to create a list of transitions to be implemented in order to reach a couple of goals and abilities. These are typically identified as drivers based on the company strategy.

In order to reach this goal there are a couple of sub-targets. Although this is an iterative process, it's good to know the purpose and goal of each phase (sub target).



What is our strategy? - What are the drivers?

The first step to set the course is to define the drivers for the need to scale. Drivers are the reasons why we need and want to scale our software, and are often closely aligned to the company's strategy. Typical drivers are a desire to enter a new market or market segment, or to become more innovative. Rapidly expanding organizations can also suffer from growing pains, for example when if the software architecture doesn't scale with the organization, or when well-trained staff is hard to find.

Start with the external drivers

Most drivers are external, i.e. derived from outside the company. It can be customers, markets, or global regulatory requirements that want us to make the changes. To get our brain to start formulating these drivers we can ask questions like:

- What are the market or customer expectations on our organization?
- What are the management team expectations on our organization?
- Have any regulatory requirements been changed that we need to cope with?
- Where is the market heading – What are our competitors doing?

Utilize the help from the structure of the "driver groups" and ask what really drives the compa-

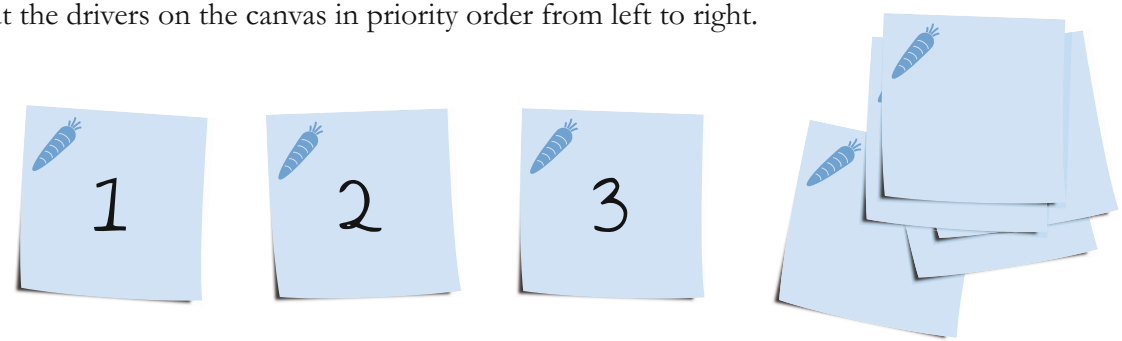
ny to a successful future. Is it the OPEX cost that needs to be lowered or is it rather a complete new business model that will outperform our competitors?

Fill the top area of the canvas with blue post-its, each with a driver. This can be done in many ways. For example, we can let the participants do this one by one. We can also do it in small groups and ask one person from each group to present and argue why the drivers are important to have from an external point of view. It's important that all post-its are discussed and understood (and maybe even rephrased) by everybody before being put on the canvas.

When all notes are on the canvas we most likely need to group and reduce them according to a prioritization.

- Group them in new common areas if possible as the number of post-it notes grows.
- What are the two to three most important items to focus on?

Put the drivers on the canvas in priority order from left to right.



Good things we want to keep

The drivers primarily capture things we want to change in order to scale. However, at this point it can also happen that there are proposals that at first glance seem good but at the later discussion turn out to have negative impact on already existing (good) drivers. Here we have the possibility to remember this by creating post-it for a driver we want to keep. Keep these post-it notes to the right on the driver's area, to distinguish these as external drivers.

www.dbooks.org

Continue with internal drivers

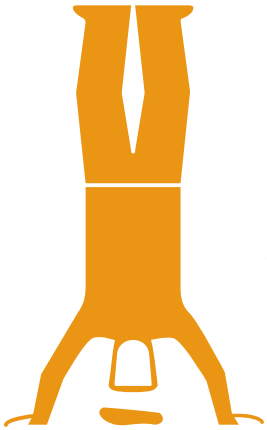
Most drivers are external, but sometimes we also have internal drivers. These are things the customer or market not specifically ask for, but we still want to achieve as we think it is the right direction for the future. Ask questions like:

- What would make us proud?
- What kind of work climate do we want to have?
- What do we need to be well prepared for the future (which is sometimes not so easy to predict – so why not become as prepared as possible)?

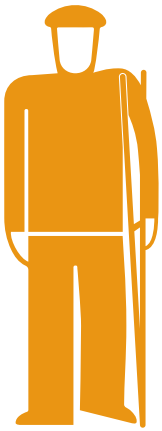
Generate blue post-it notes. Explain, discuss and prioritize them. Add the most important to the left of the external drivers. Post-it notes that are removed due to prioritization can be stored in a "parking lot" for later retrieval if needed.



Desired abilities and current inabilities



Now when we have all the blue post-it notes in place, next up is to identify the abilities. Abilities are aspects of the software development such as cost, performance, and quality, which are possible to measure without knowing any details of how the development is made. In the workshop the goal is not to create all-covering measurements of the entire software development, but we will focus on abilities we need to have in order to meet the drivers, the desired abilities. We will also identify the current inabilities, the abilities we have today that we need to improve in order to get the desired abilities. Start by looking at the drivers and formulate measures related to them. If the drivers are about customer satisfaction, and product quality, the desired abilities



will most likely be about number of issues, customer satisfaction survey results or time to market. In a company works with KPIs, hopefully we will find our Drives in these.

The desired abilities will be the "Definition of done" in the implementation phase, so look carefully at them and ask, "Have we succeeded if we reach this?".

When the desired abilities are in place continue with the current inabilities. What is it that is not good enough – what are our growing pains? Obviously, many of them will be similar to the desired abilities, like for instance customer support availability: a desired ability is 24/7, but current inability is office hours.

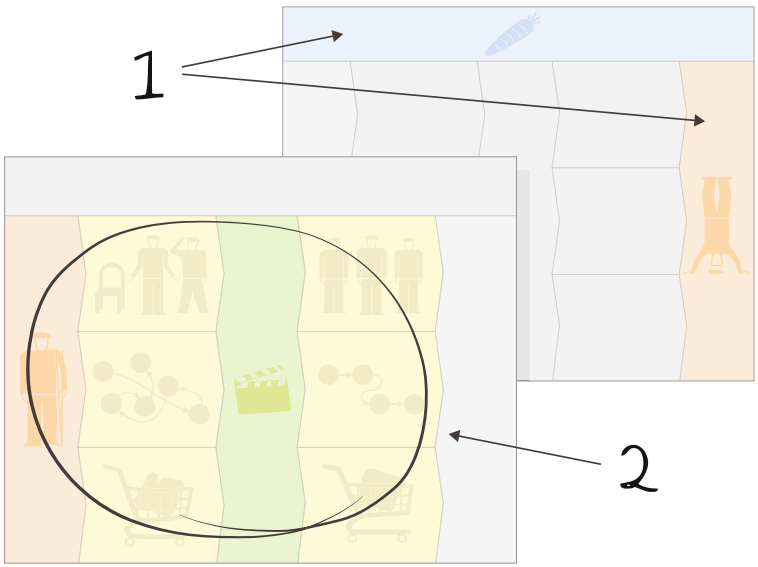
Put up post-its both for the desired abilities and the inabilities and make sure they cover all the drivers.



Iterative process

To identify drivers, desired abilities, and inabilities is an important sub target. It often requires an iterative workflow (considering the software development as a black box) that doesn't finish until we are on common ground with the company's strategy and vision.

In many companies, the people that define drivers and abilities, and the people that can break these down in terms of organization, process and product, are not the same. In such cases, it's possible to first have a limited workshop with management only, to decide on drivers and identify abilities, and use this outcome as input for a subsequent workshop .



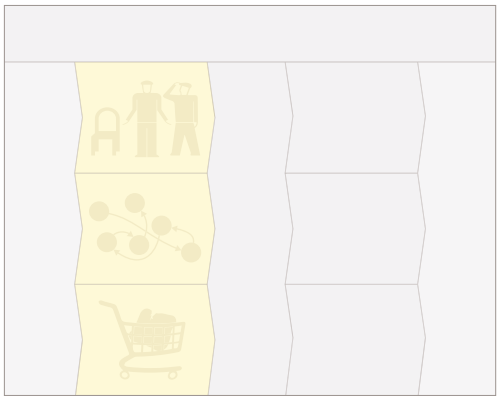
www.dbooks.org



Explain current abilities with domain capabilities

Now it's time to open the black box. We need to understand why we have inabilities, we need to analyze and describe our current situation as is.

Ask challenging questions within all three domains: product, organization, and process. Identify the current domain characteristics that potentially are the problems causing the inabilities. Find characteristics, that if changed will solve or remove the current inabilities. A domain characteristic is simply explained as a hallmark for the company's software development. Examples of such are "manual test and delivery," "no routines for source code management" and "no process for customer requirements."



Don't just look for the no-brainers, we might have to nest and ask "why" for quite a while to find the root cause. Why do we have a manual test? Maybe is it because we lack the competence to create automatic tests. If so, this is a yellow note in the organization domain rather than one in the process domain – or both with a line in between depicting the relationship.

During this phase we really need domain experts, people from the company that know how things actually work. What are the actual practices that are used? What does the product architecture look like and what affects on the abilities does it have?

Troubleshooting requires deep knowledge about dependencies - causes and effects. This means we also need people with good general knowledge about the domains and the characteristics of different solutions. If the knowledge can't be found within the company, bring in external competence to the workshop. Such people can also assist as moderators and help getting a holistic view of the discussion.

Use the building blocks within the domains to, in a structured way think through how we actually work and why. In the organization domain, go through all four blocks (structure, culture and leadership, people management, and improvements) to search for reasons to the inabilities. Let the experts from this domain shortly explain the current characteristics of each building block. Discuss if we should add a yellow note.

Use competence from all three domains to really find the reasons to the inabilities, what prevents us from reaching the desired abilities. Put up yellow notes and draw arrows between them to show dependencies.

Don't stop until all inabilities are explained by at least one domain characteristic. For non-obvious relations, draw an arrow in the canvas, from the yellow domain characteristic to the orange inability.

We might identify existing characteristics we're about to remove or change, but during the discussion realize we ought to keep these. Leave these in the canvas and mark them with "keep", to remember they are also part of the desired solution. These won't have any arrows to current inabilities.





www.dbooks.org



Find a solution

Now the creative part starts. We need to decide on what changes to make in order to improve and meet the abilities and drivers. In other words, we need to define the transitions the company needs to do.



The goal is to have yellow post-its in the three software domains fulfilling the desired abilities. Each yellow note in the as-is domain should have a transition explaining how it is transformed into the desired solution. When not obvious, draw arrows depicting the transitions. Similarly, all yellow notes in the desired domain should have corresponding transitions needed to have them implemented.

In reality, this isn't so easy. Most likely there will be many alternative transitions with different pros and cons and no obvious "silver bullet" solution. Also, a transition in one domain may also affect other domains. As an example, to make a quite obvious change to a process might also call for changes to the organization, e.g. new skills needed. In these cases, we need to add yellow notes also explaining the needed "supportive" desired domain characteristics.

Make sure to evaluate different transition possibilities. Also make sure that all desired abilities have been addressed before the final set of transitions is defined.

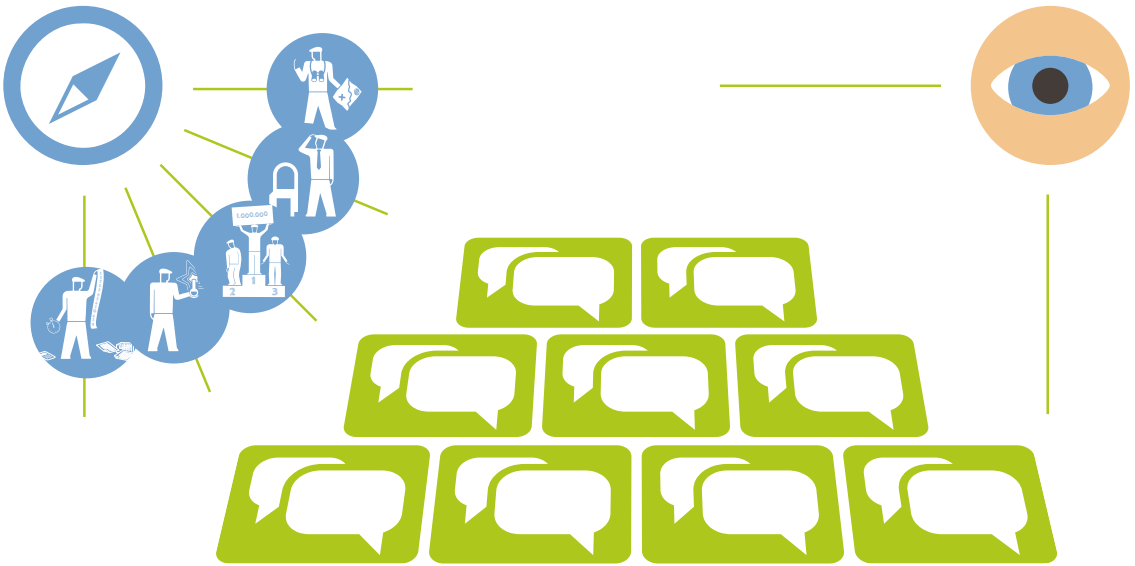


Use the body of knowledge and experiences to be creative

Although both finding the root causes and searching for the best solution are a very creative tasks, it is also here we can use this book to utilize the experience from previous situations in other companies. Journeys and travel stories provides us examples and captured experience from other companies who have been through similar situations.

The best way to find relevant scenarios is to look at your drivers. In which one of the five main driver groups do you see your drivers. Then, go to the "The compass" part of the book to find relevant scenarios to read for your driver group.

An alternative way of using the book is if you want to know how companies similar to yours have done. Browse through the journeys and see if you can find a company relevant for you. As every journey is exemplified with several Travel stories, we encourage you to read the entire journey for the selected stories. The canvas for each journey and Travel story gives a quick overview in order to determine if a more thorough read is needed.

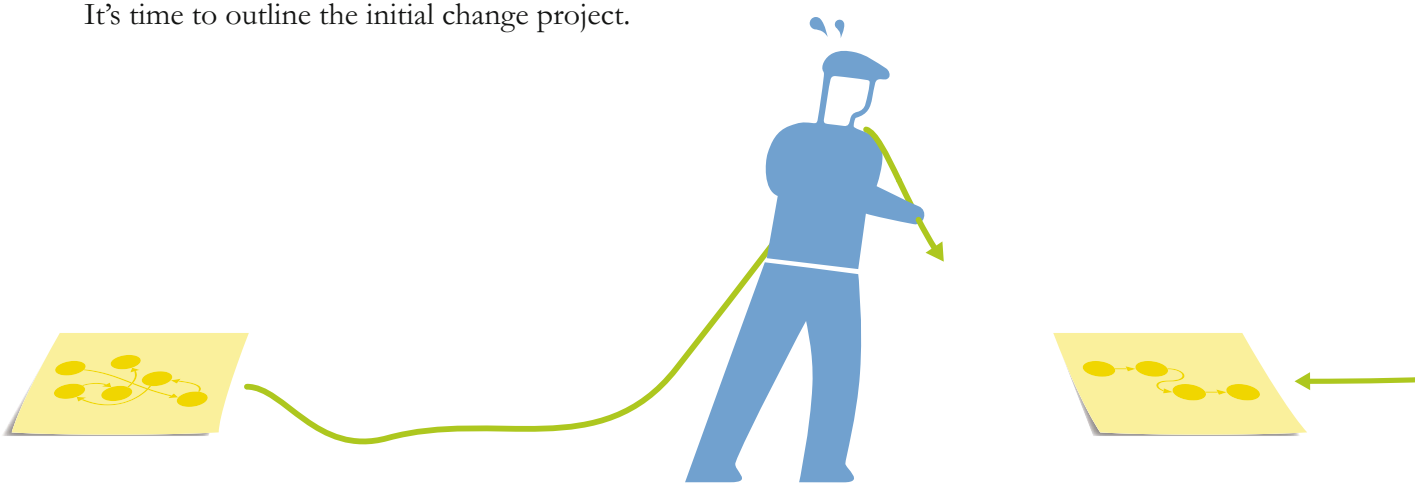


www.dbooks.org



Decision time

All sorts of design ideas have been discussed, pros and cons argued, and trade-offs worked out. It's time to outline the initial change project.



When the characteristics of the future software model have been defined and we know what changes we need to make, document the changes. Start with the main transitions and draw arrows to show how the new domain characteristics result in the desired abilities.

As we know we might need additional transitions to create the pre-conditions needed for the main transitions, add also these to the canvas and draw arrows to show how they support the main characteristics.

At last, if we need to execute the transitions in any specific order, also add notes or arrows that describe the dependencies between the transitions.

Use the dependencies between the transitions and put together an ordered list of transformations. This list can be used throughout the implementation of the change. In the next chapter you will see how to succeed with the actual change implementation phase.


Done - let's get on
with the real work

The real work



... most organizations lack all the skills needed to implement and optimize business processes ...

Successful process management requires an agile iterative approach to process change.

Gartner Inc.



The Scaling Management Framework will help you to identify the transformation journey to take. It can also help you in dividing the transformation project into reasonable steps. While understanding what to do is crucial, it's not the same thing as knowing how to do it. Every change needs success criteria, a project team, a plan and a budget. However, the planning accuracy is not saying if your change project will be a success or not.

A successful implementation requires a lot of work from all employees involved. Everyone needs to understand why there is a need for change. They have to acknowledge the reason that drives the change and understand how it affects the organization and the ways of working. If it isn't clear "what's in it for me," there is a risk that the project will meet resistance from the employees.

One obvious factor for success is the ownership and the attention from management. High-level progress should be communicated from top management and not only from the project leaders, which should focus on more detailed information sharing. Visibility and transparency in the change process is a key aspect to grow trust and motivation among the employees. The management team should also support the project removing any impediments that might arise during implementation.

A change project can be set up similar to an agile project. Using a prioritized change backlog, use small iterations and retrospectives to minimize the work in progress and keep the lead-time short. The agile change process provides a lot of tools to follow up the results and track the progress. All measurements should help to drive the change (or at least not slow them down) and serves also to reinforce the motivation.

www.dbooks.org

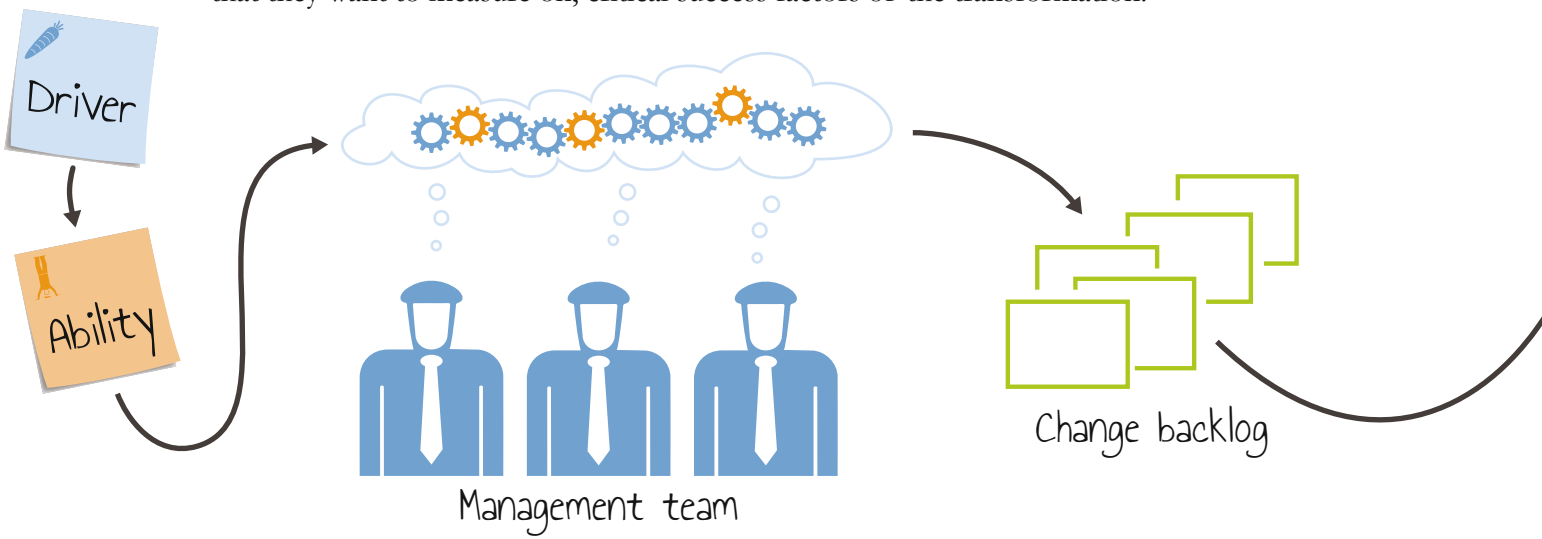
Agile change center

The Agile change center is a framework for guiding any transformation, based on the same principles that guide agile development. Through determined commitment to small, continuous and incremental change, the Agile change center may be used to investigate, propose, facilitate, execute and deliver any of the change scenarios presented in this book.

The Agile change center primarily seeks to accelerate the transformation and provide:

- Organizational alignment around agreed drivers and abilities (KPIs)
- Management engagement in the transformation
- Mechanism for continuous and sustainable improvement aided by visibility, feedback and reflection

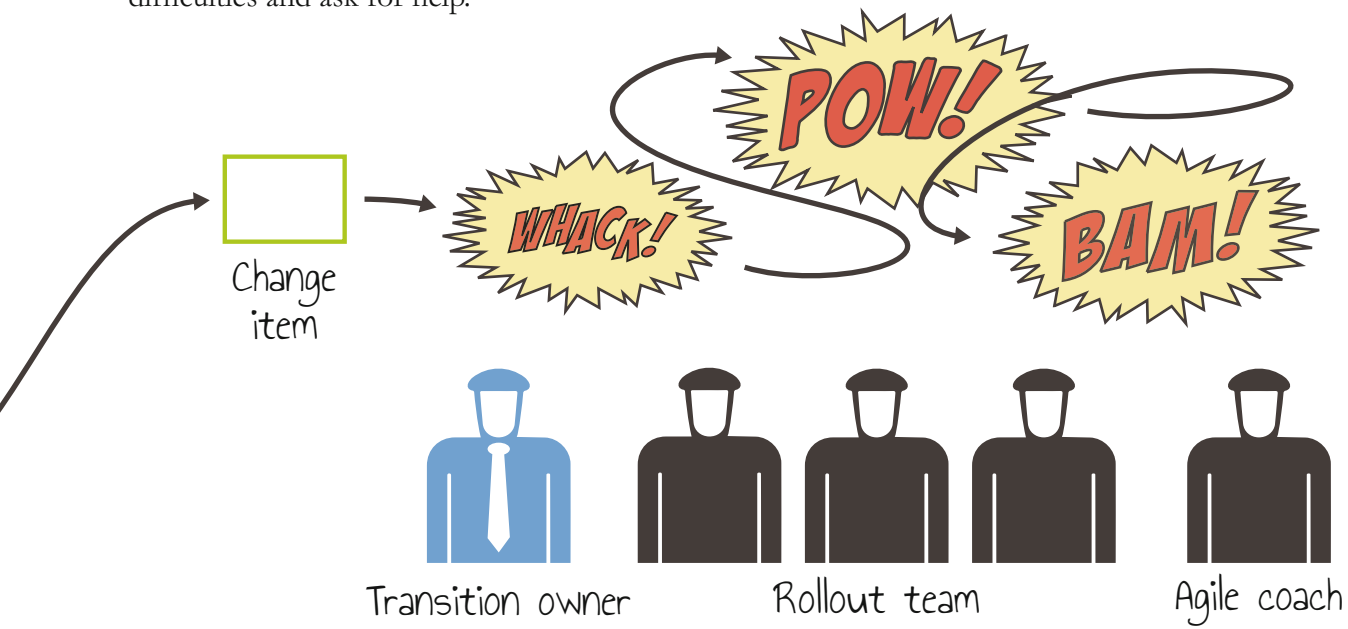
The center consists basically of two teams, the management and the doers. The management team defines drivers and prioritizes and aligns the overall transformation. They set all drivers, such as goals based on sales growth or return of investment. They have also identified abilities that they want to measure on, critical success factors of the transformation.



The implementation teams are manned with all necessary competences to roll out the transformation. Two important members of such a team is the transformation owner, who represents the management team, and the Agile coach, who facilitates the performance, learning and development of the individuals in the team, as well as the team per se.

The management team meets every 2–4 weeks, to refine the change backlog, prioritize change items and to follow up on previous change items. The change backlog contains all actions in the transformation. It's basically a list of change Items that is prioritized by the management team. A change item can be any opportunity for improvement, in any of the three SMF domains. The opportunities can be expressed as impediments of a problem, investigations, proposals or suggestions.

The implementation teams meet every 1–2 weeks, for iteration planning and commitments, to review progress and give feedback and to look back at changes that already have taken place. All but the transformation owner meet as well every one or two days to synchronize work, raise difficulties and ask for help.



www.dbooks.org

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org



© The Author(s) 2017
B. Fitzgerald et al., *Scaling a Software Business*,
DOI 10.1007/978-3-319-53116-8



www.dbooks.org

Alphabetical list of terms and definitions

Abilities	Abilities can be seen as Key Process Indicators (KPIs) for the transformation journey. The KPIs are preferably the ones used in the company's Balanced Scorecard. Abilities have to be measurable in order for us to recognize if we're getting any better. The whole organization, with its three domains, is considered being a black box when we measure such abilities.
Case study	A case study gives real world experience from a company that is in the midst of or past their digital transformation. Some companies have been successful and some have not.
Characteristics	The characteristics of an organization is how it looks when we open up the black box and see in detail what is happening. The current characteristics are the root cause of the inabilities and the desired characteristics are the reasons to why we achieve the desired abilities.
Current organization	Defines the root causes of an inability within the organization domain.
Current process	Defines the root causes of the inabilities within the process domain.
Current product	Defines the root causes of the inabilities within the product domain.

Digital transformation	The process of radical change, where companies are converting from an analog to a digital world.
Digitalization	The process of converting from analog to digital. Digitalization means a shift in focus from products, hardware, and mechanics towards software and services and possibly disruptive business models.
Drivers	The key priorities a company's management board would look for the software organization to address to cope with the digital transformation. They can be grouped into five main categories of drivers and equals to the goals of the software organization. These drivers are the rationale for why we need to change the software organization.
Inabilities	Inabilities are the abilities that currently hinder us from achieving the drivers. The inabilities are mostly visible outside of the organization, by other entities within a company or outside a company such as by customers.
Organization domain	The organization domain includes all organization and business processes including, but not limited to, how to structure the organization, culture and leadership, people management and how to drive improvement work.
Process domain	The process domain covers all aspects of how a product or a service is developed and tested. We have chosen to divide this into two subcategories: engineering and project management.

www.dbooks.org

Product domain	The product domain covers the products and services that we offer on the market. This domain deals with aspects like the software architecture, how we structure the product or service, the infrastructure and our distribution channels.
Scale	Software companies have to scale, which is another way of saying change with the digital transformation.
Scenario	A condensed set of lessons learned extracted from case studies with similar drivers. A scenario is lessons learned by several companies with hands-on experiences from similar digital transformation.
SMF	The SMF (Software Management Framework) helps companies with one of the key challenges of European Industry: how do we transform our organization when software is becoming a critical part of our offering and asset? The digital transformation that comes can partly be driven by the technological evolution and partly by the business. The SMF is distinctive in the sense that it explains the transformation in three domains – organization, products and processes – in the same model. The SMF consist of the map, the compass, the travel brochures, and the travel stories.
SMF canvas	A graphical tool for understanding and describing the transitions needed for the software organization to carry out a digital transformation.

Software organization	A software organization can be an IT department and/or a software R&D department.
The compass	The five common drivers are used as the compass in the transformation journey. They will guide you through the map and help you to pin down your own digital transformation journey.
The journeys	A database of industrial best practices and tools to support enterprises in their digital transformations. The database contains travel stories and travel brochures to be used at the digitalization journey.
The map	The SMF canvas is used as the map of the transformation and it helps in creating a digitalization strategy.
The travel brochure	A scenario can be seen as a travel brochure for the journey.
The travel story	Case studies give the reader the concrete travel stories.
Transformation	A process of radical change that orients an organization in a new direction.
Transition	One or more transitions are needed to make the digital transformation in a company. A transition is the key activities needed to go from the current characteristics to the desired characteristics. In the SMF canvas, this is graphically represented as going from one or several yellow post-its on the current side to one or several yellow post-its on the desired side.

www.dbooks.org

Desired abilities	The desired abilities are the desired state that we want to reach for our KPIs. They are clearly defined and are measurable. They will indicate if we have reached our goals with the transformation performed or not.
Desired organization	Defines the desired characteristics of the organization domain, which enable us to achieve the desired abilities, which turn is the end goal for our drivers.
Desired process	Defines the desired characteristics of the process domain, which enable us to achieve the desired abilities, which turn is the end goal for our drivers.
Desired product	Defines the desired characteristics of the product domain, which enable us to achieve the desired abilities, which turn is the end goal for our drivers.
Your journey	When a company uses the map, the compass, the travel brochures and the travel stories to define their own digital transformation journey.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org