

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

# scaling a software business

The Digitalization Journey

 Springer Open

# Scaling a Software Business

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

# Scaling a Software Business

The Digitalization Journey



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 Computing Institute (MAPCI)  
Lund University  
Lund, Sweden

Henrik Cosmo  
Mobile and Pervasive Computing 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](http://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](http://www.lero.ie)).

# 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 EU-REKA 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.



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.



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.

## Co-created by industry and academia professionals



**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.

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.



**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.



**Ove Jansson**

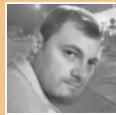
Mon Petit Studio

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.



**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

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



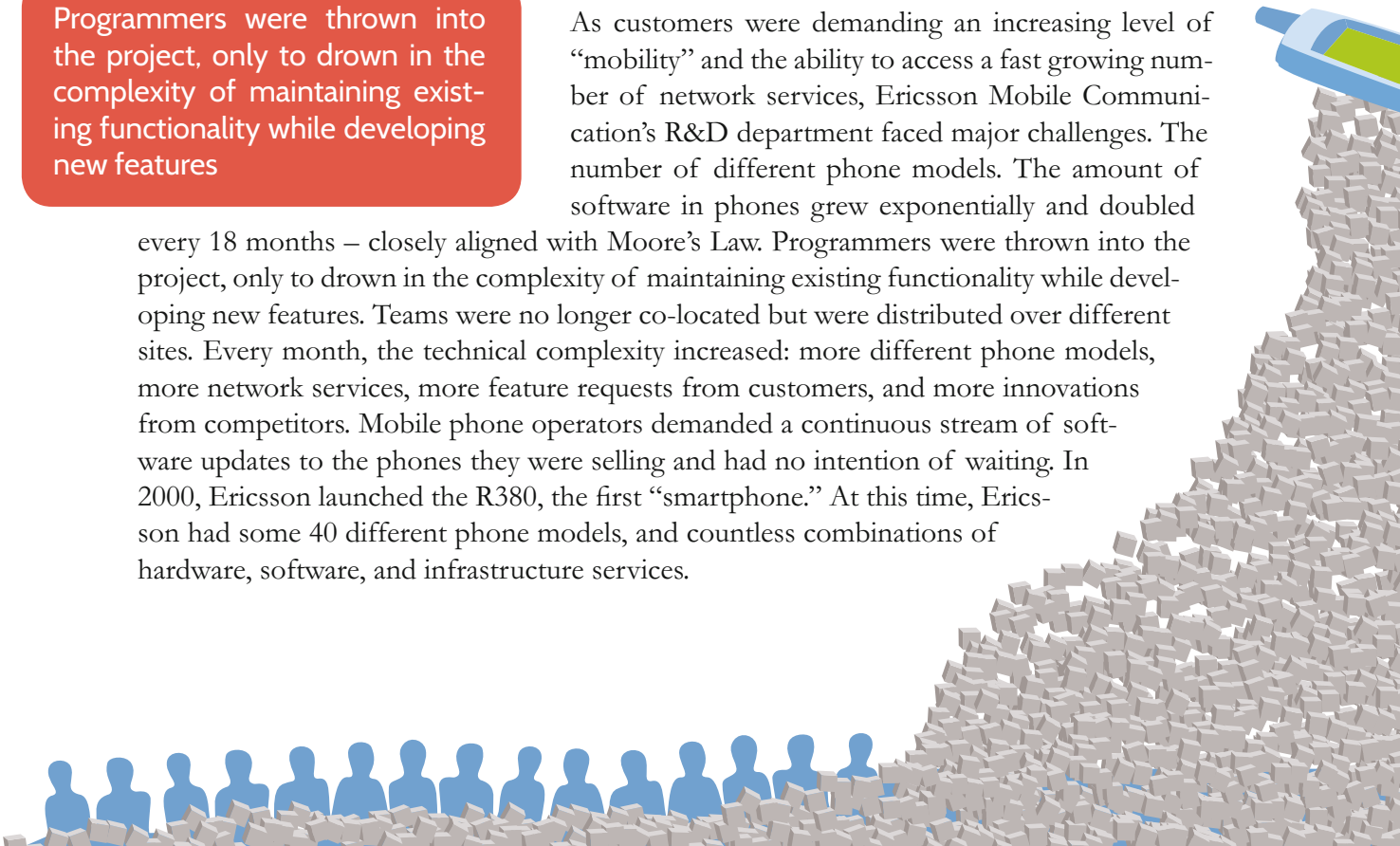
technical and telecoms-related platform software. Only one of the developers was responsible for the phones' user interface. You could use the phone to make calls – but that's about all you could do.

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.

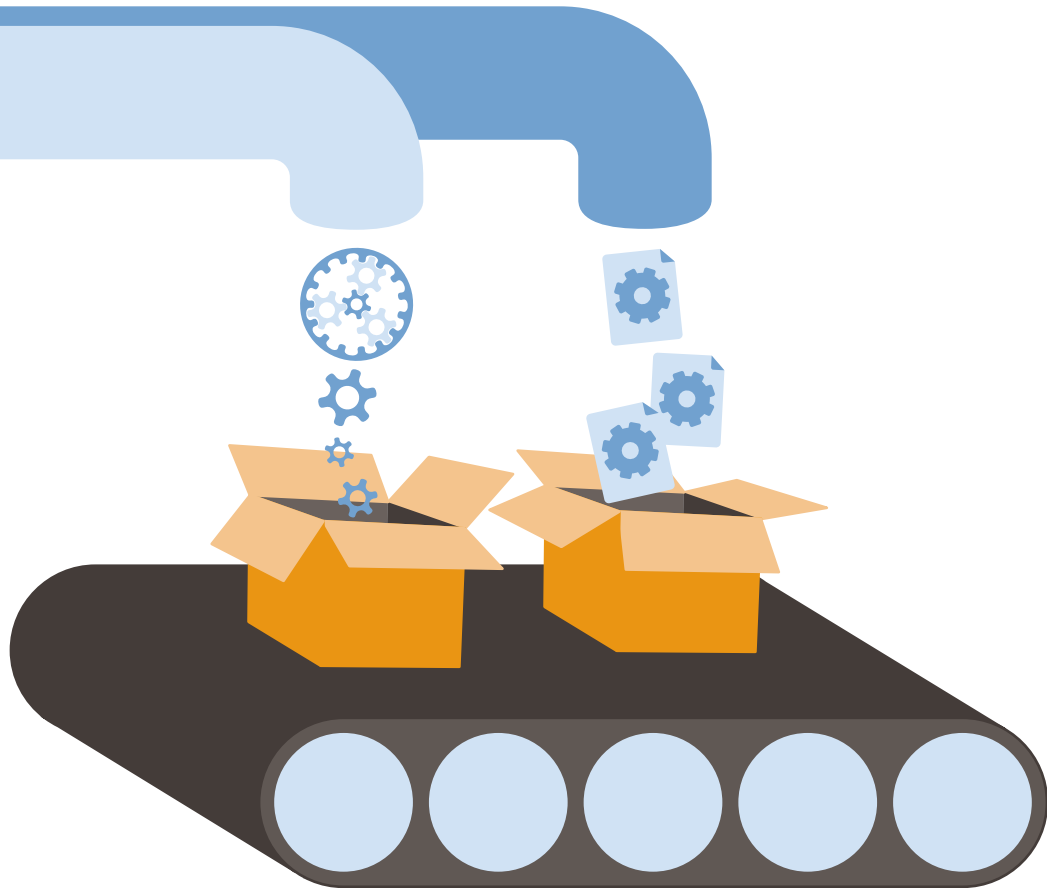


It was time to take the plunge. As the software grew exponentially, the change process had to embrace the entire organization. Product and system architecture, organizational changes, business and development processes – all had to adapt in a coordinated manner towards a common goal. Rules for what can 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.

# 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!**

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.

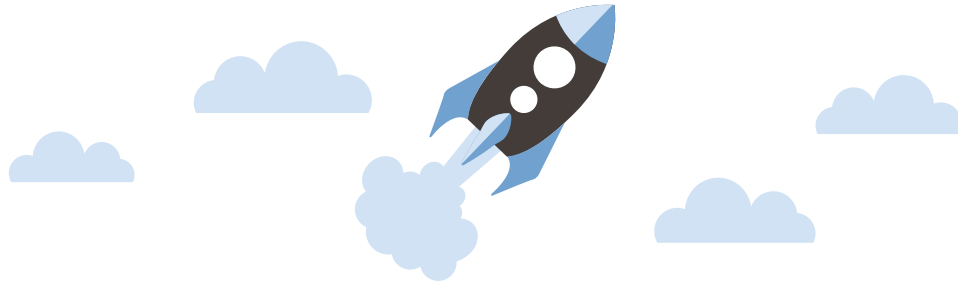
What motivates these companies to transform their businesses, what are their goals, or to put it differently, what are the business drivers? The primary answer is that software affords development

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



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.

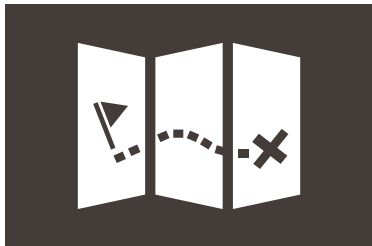
Obviously, IT is just one of the cogwheels in the digitalized business that need to spin in new ways. The entire organization will need a lift. As this chapter proceeds, it will become clear that it all boils down to a company's motivations to transform: the business drivers.

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.

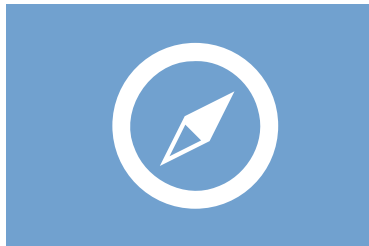
This book offers a method that will help your organization in three ways:

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

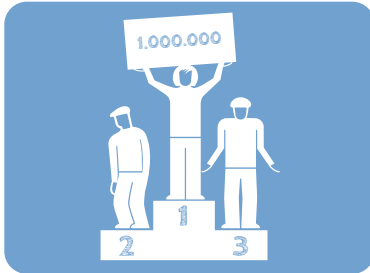


The journeys. An experience database with best practices and lessons learned from past digitalization transformations.

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.

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



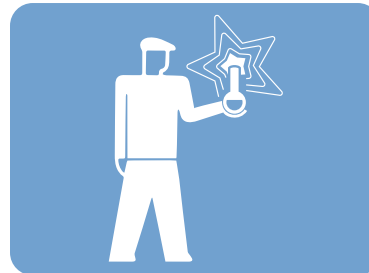
Increase quality, make operational expenditure savings and shorten time-to-market



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.



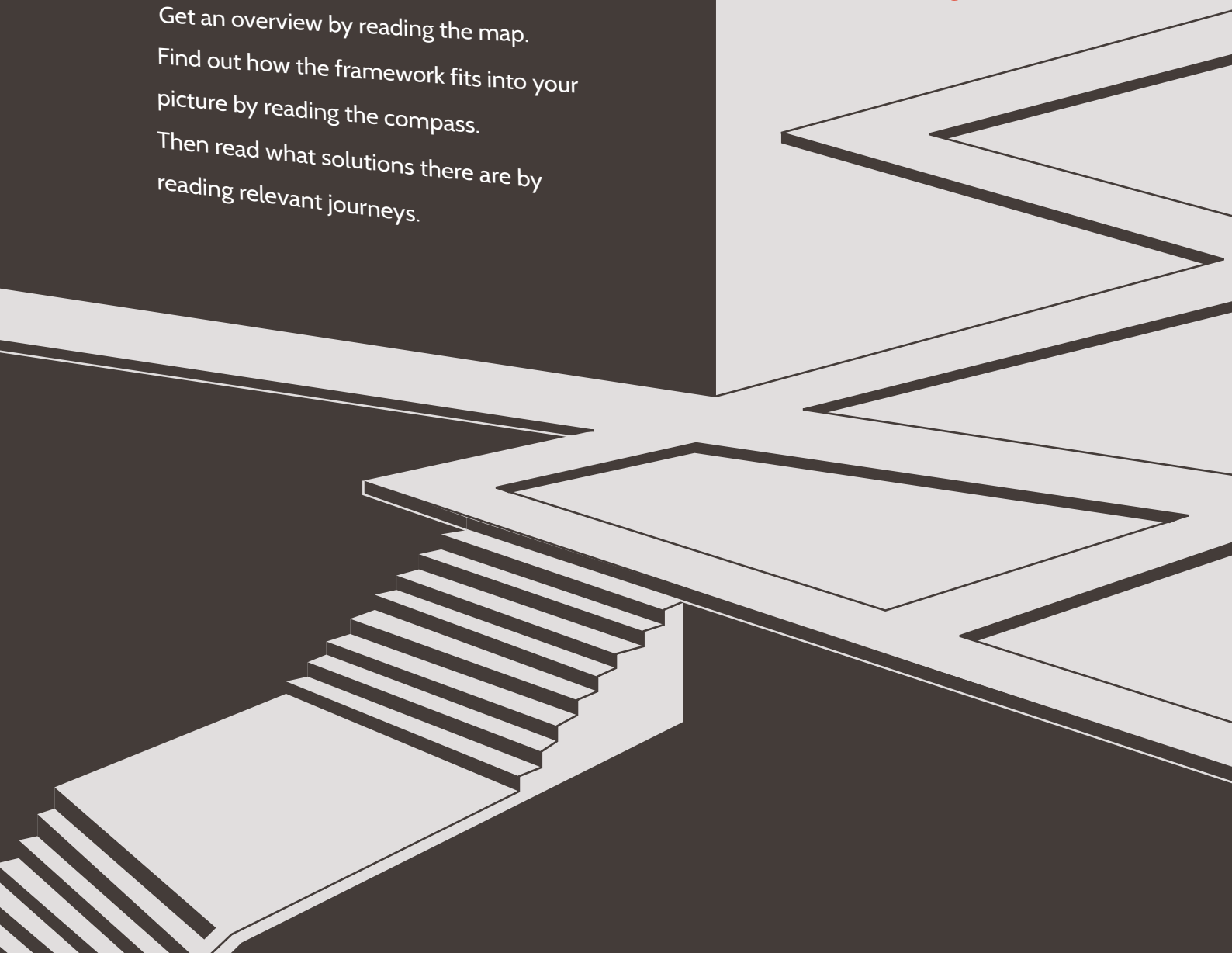
# Not to waste time

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

# The

## Scaling Manager

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



# map

ment Framework



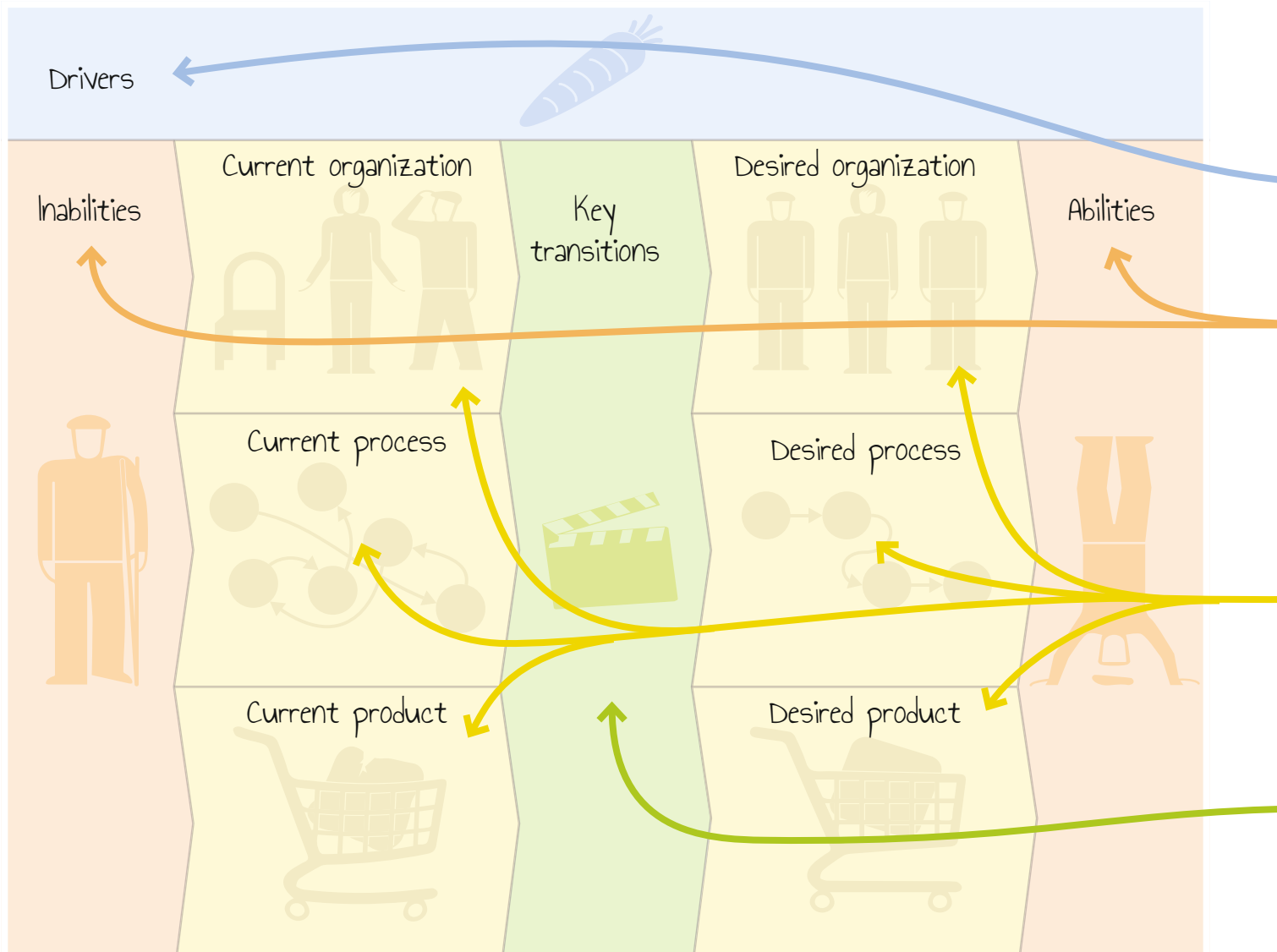
Scaling software development is a complex enterprise that can be organized in a number of ways. Since the early days of computing, hundreds, if not thousands of software development methods have been proposed. What is becoming increasingly clear, however, is that the software development function affects 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.

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.

The Scaling  
Management Framework  
covers transformations  
in three domains



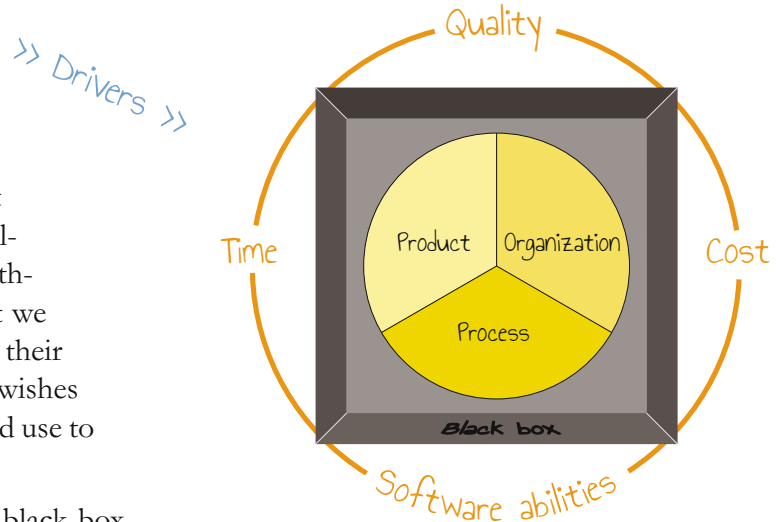
The SMF offers a multi-dimensional view on the software-scaling phenomenon. The SMF canvas, an integral part of the SMF, is a tool for understanding and describing the scaling of software development. It's designed to be visualized on a whiteboard, along with post-it notes. To the left, we have the present,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

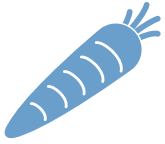
Next we observe the software development as a black box, focusing on performance, quality and other aspects that can be measured without knowing how they work. We know what we spend and how much we generate from their work. These are basically the complaints and wishes of our top management, matters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



## Drivers



The drivers of the need to change.

## Inabilities



The inabilities that make the transformation challenging.

## Desired abilities

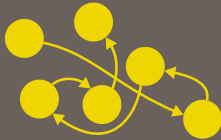


The measurable definition of done.

### Current set-up



### Organization domain

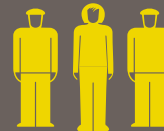


### Process domain



### Product domain

### Desired set-up



## Transition

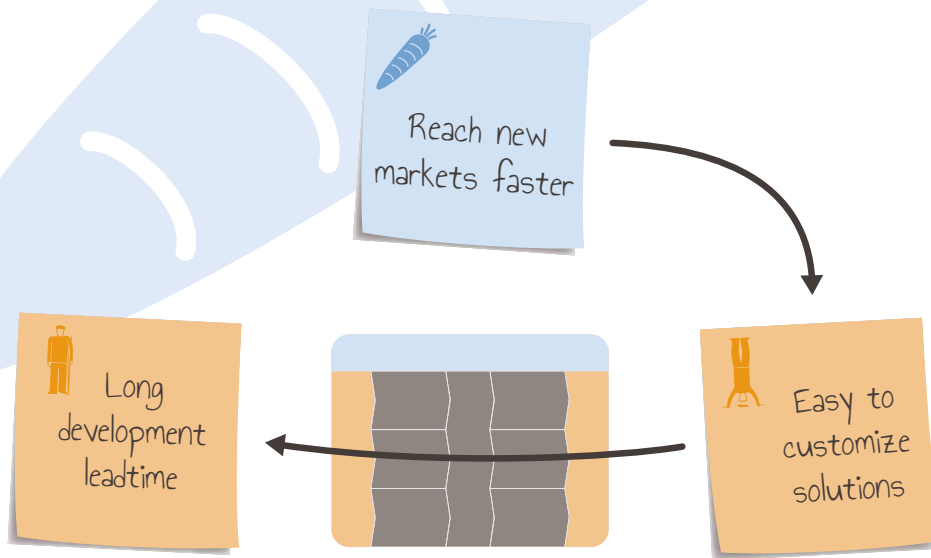


The key activities that are needed to make the transformation possible.

## 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.



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

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.

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.

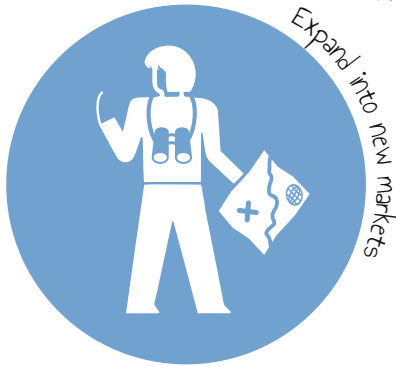




Drive revenue growth and outperform competitors



Develop innovative new offerings



Expand into new markets



Deal with leadership challenges



Make OPEX savings and improve TTM

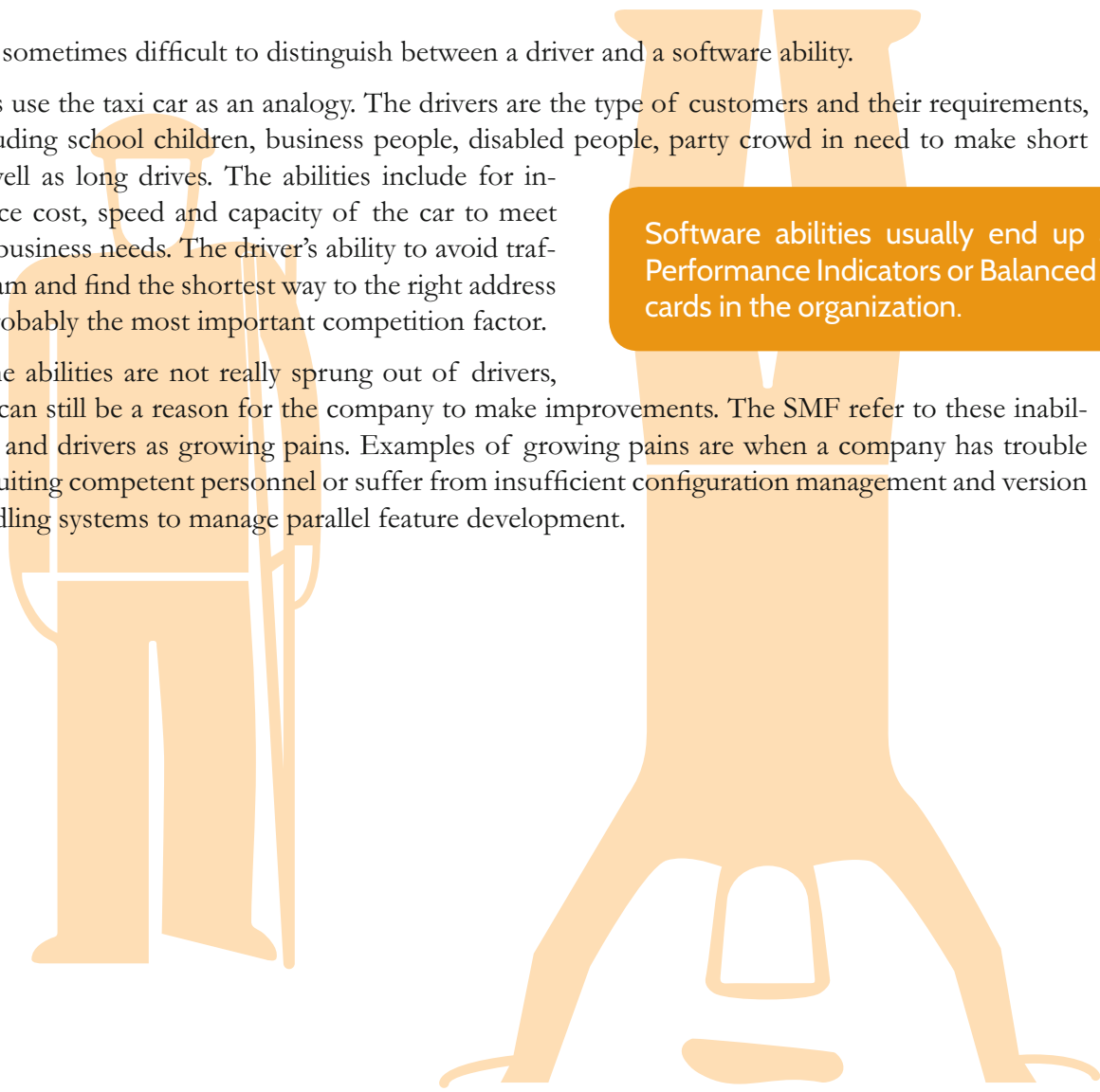
## Software abilities

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.

The background of the page features two large, stylized orange silhouettes. On the left is a person standing and using a cane. On the right is a person sitting in a chair with a prosthetic leg. The text is overlaid on these figures.

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.

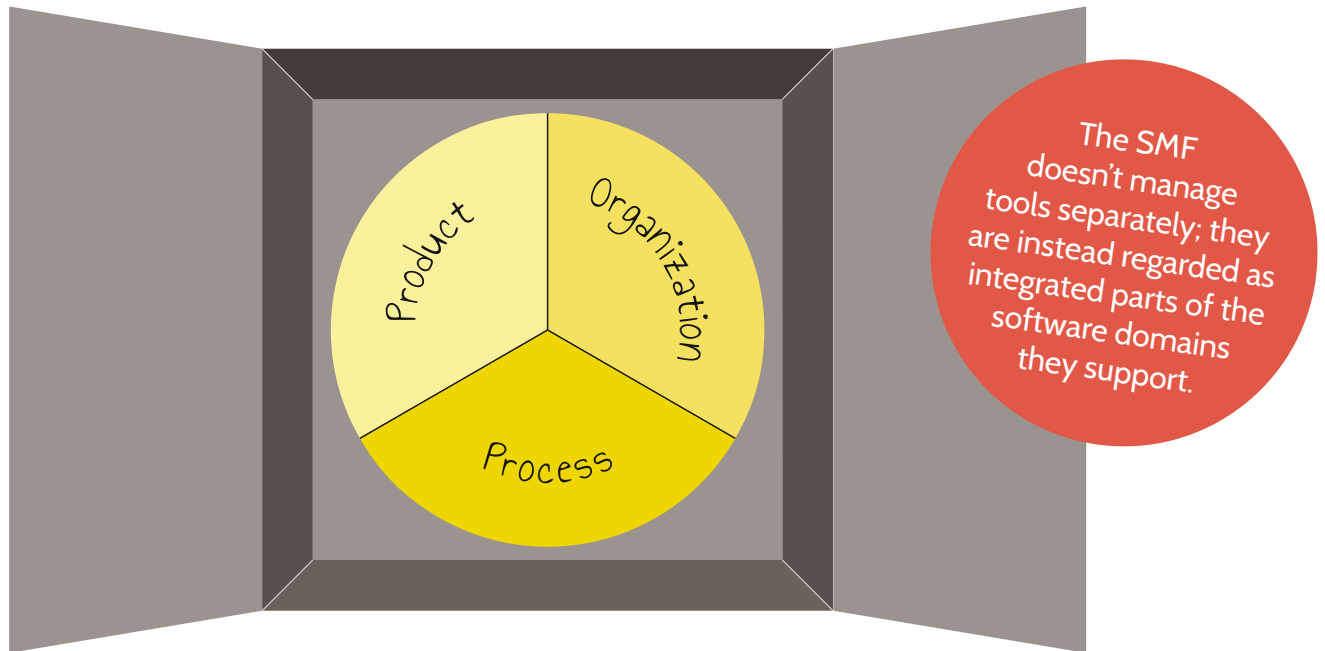
The purpose of the software model domain is to:

- 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.





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.

## 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.

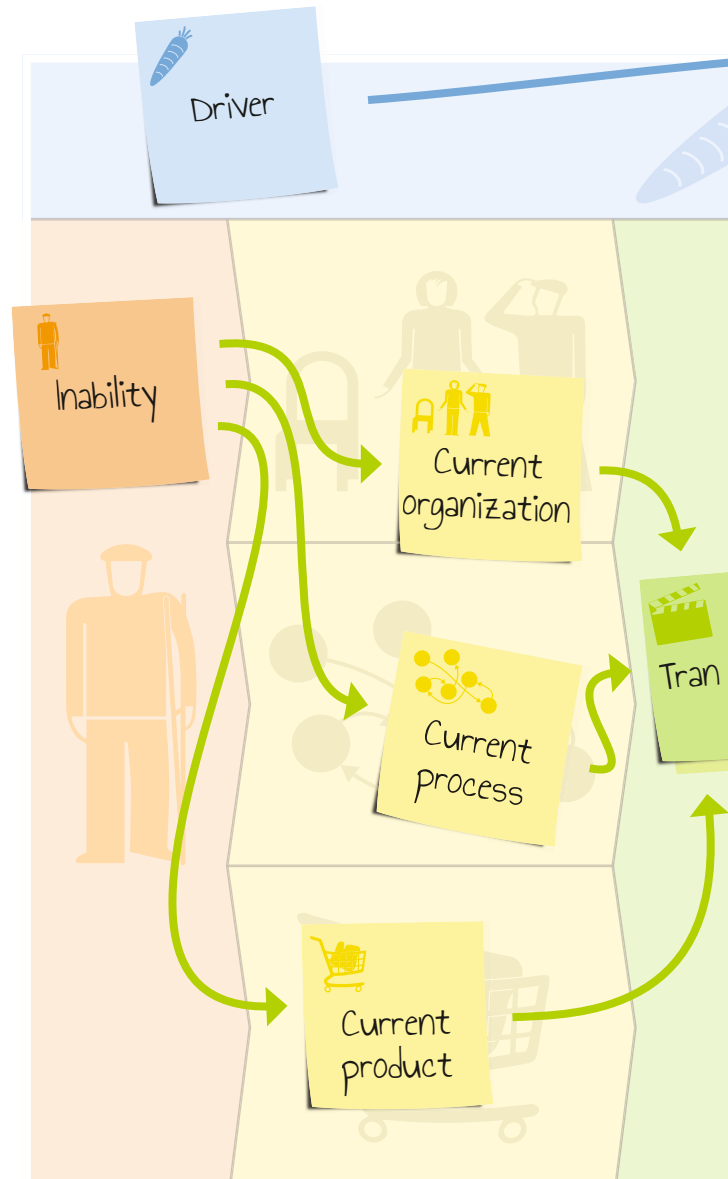
The connection between the domains, how things works in each one of them, and the current inabilities encourage us to think through all domains and their building blocks in order to understand why we have the inabilities. This is part of the self-assessment to understand what needs to be improved.

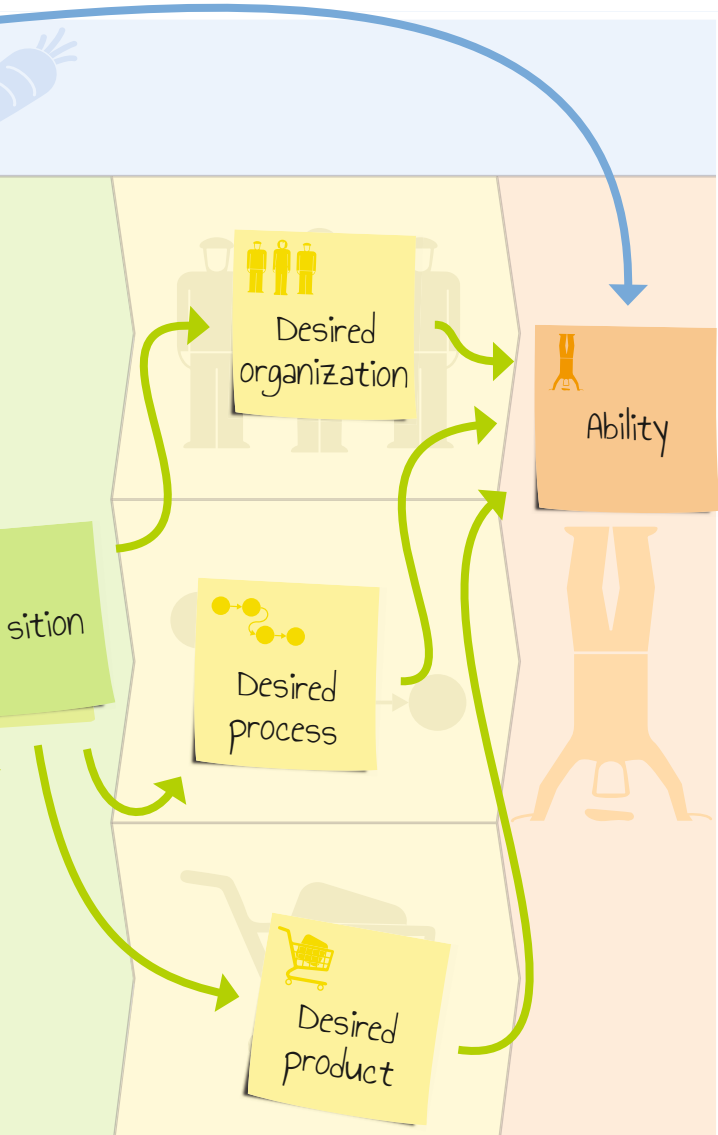
The transitions – capturing the needed change from current situation to desired situation is the most important part of the SMF. Experts within each domain will have to discuss possible changes and their impact. Yes, the SMF defines several standard change scenarios, but still most companies have to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

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.

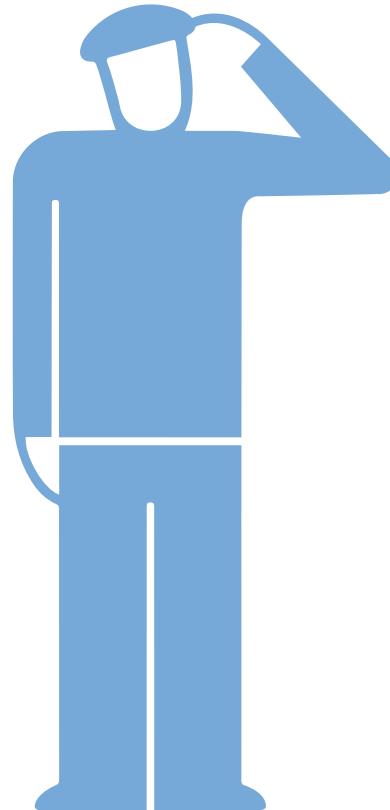
Why do we do  
what we do, as we do?

Why? Why? Why?  
Why? Why? Why?  
Why?





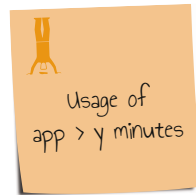
Why do we want to scale the business?  
What abilities would we need to be able to scale successfully?  
Can we measure the abilities as KPIs?



## 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.

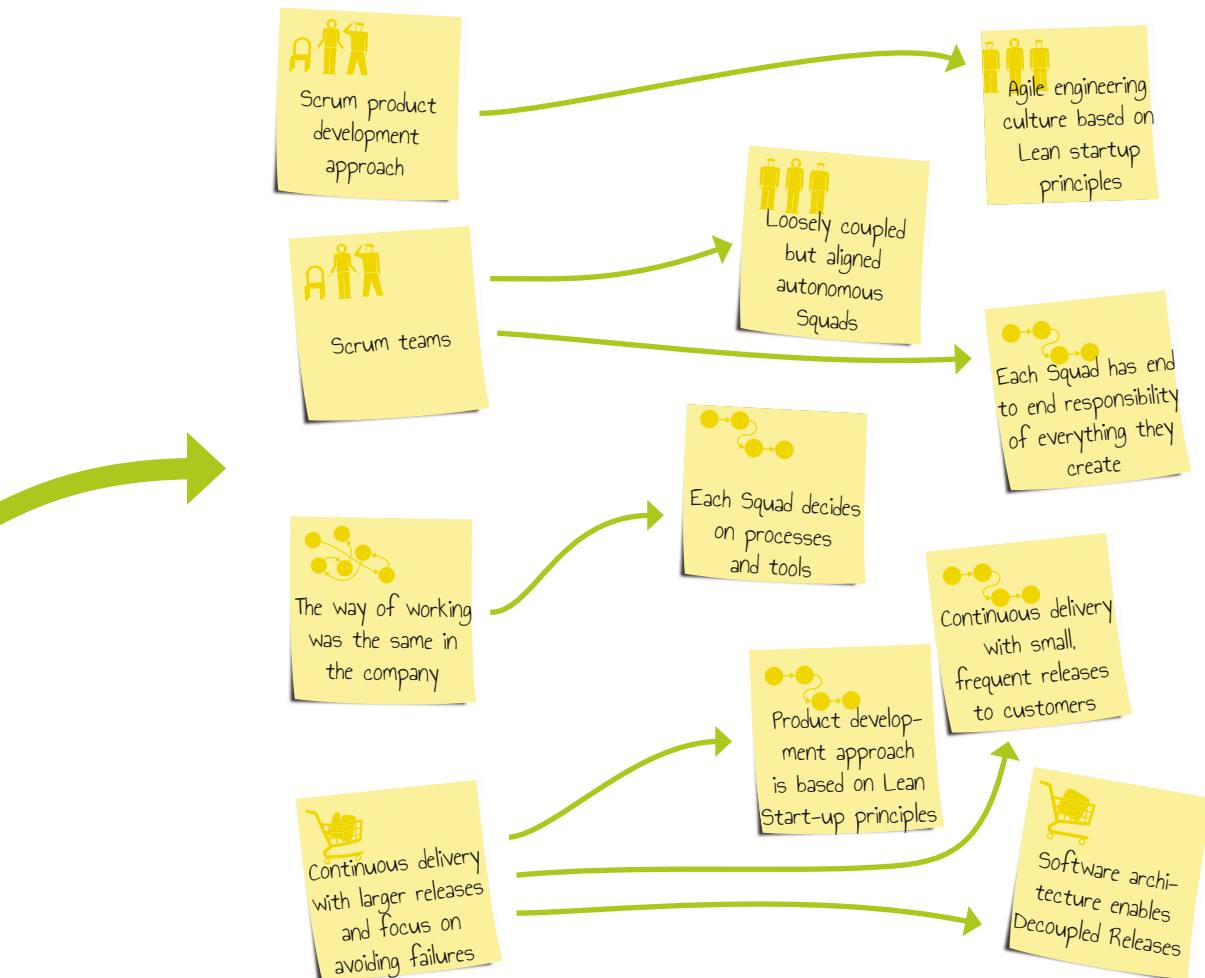
How did Spotify achieve to make the software service so successful? According to the vast amount of articles that have been written on the topic, they simply seem to have decided to “create the best streaming music service”. We can assume this was their driver. What they intended to measure was the number of Spotify streams. This was their desired ability. The higher the number of streams (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

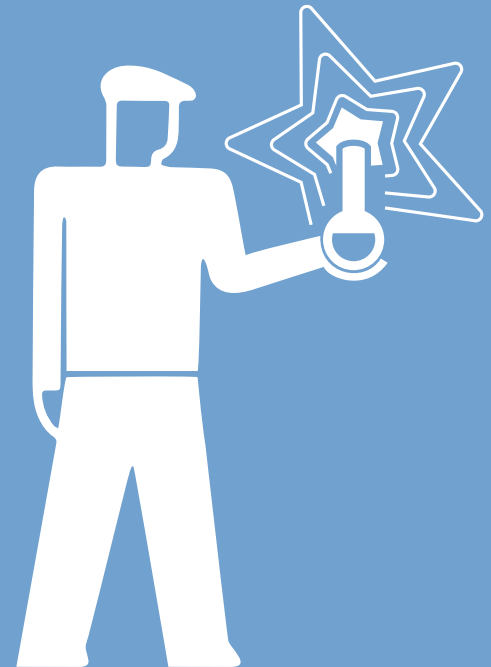
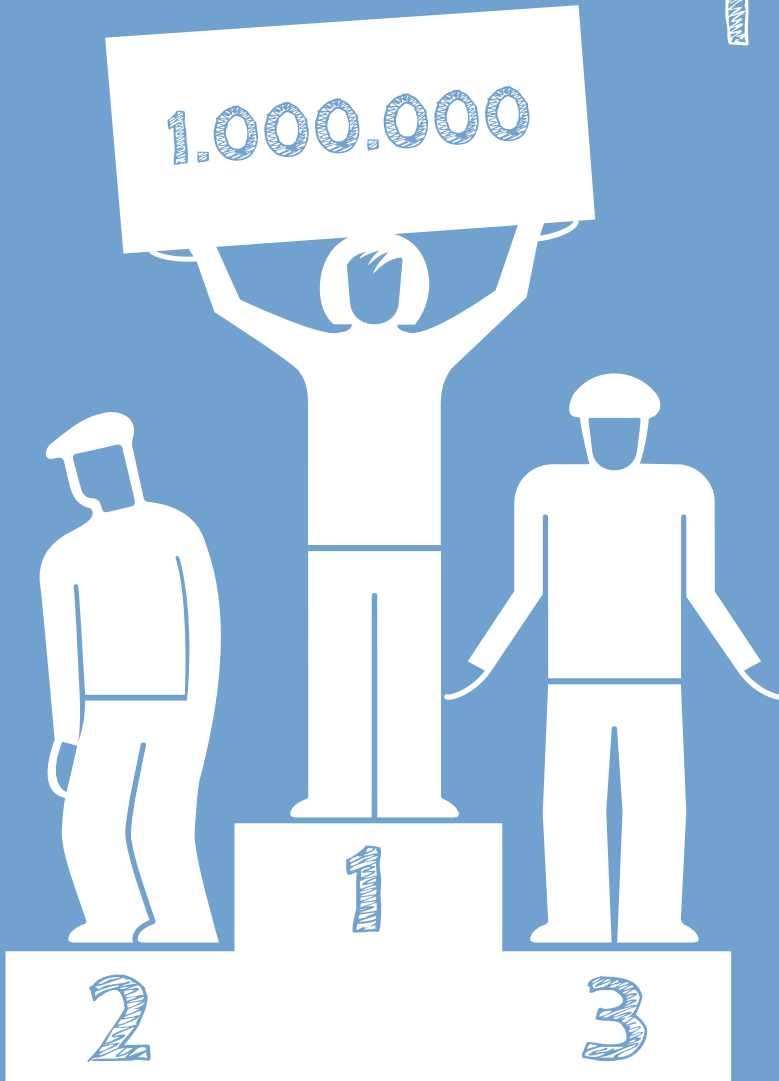
# The CO

Ways to find

# mpass

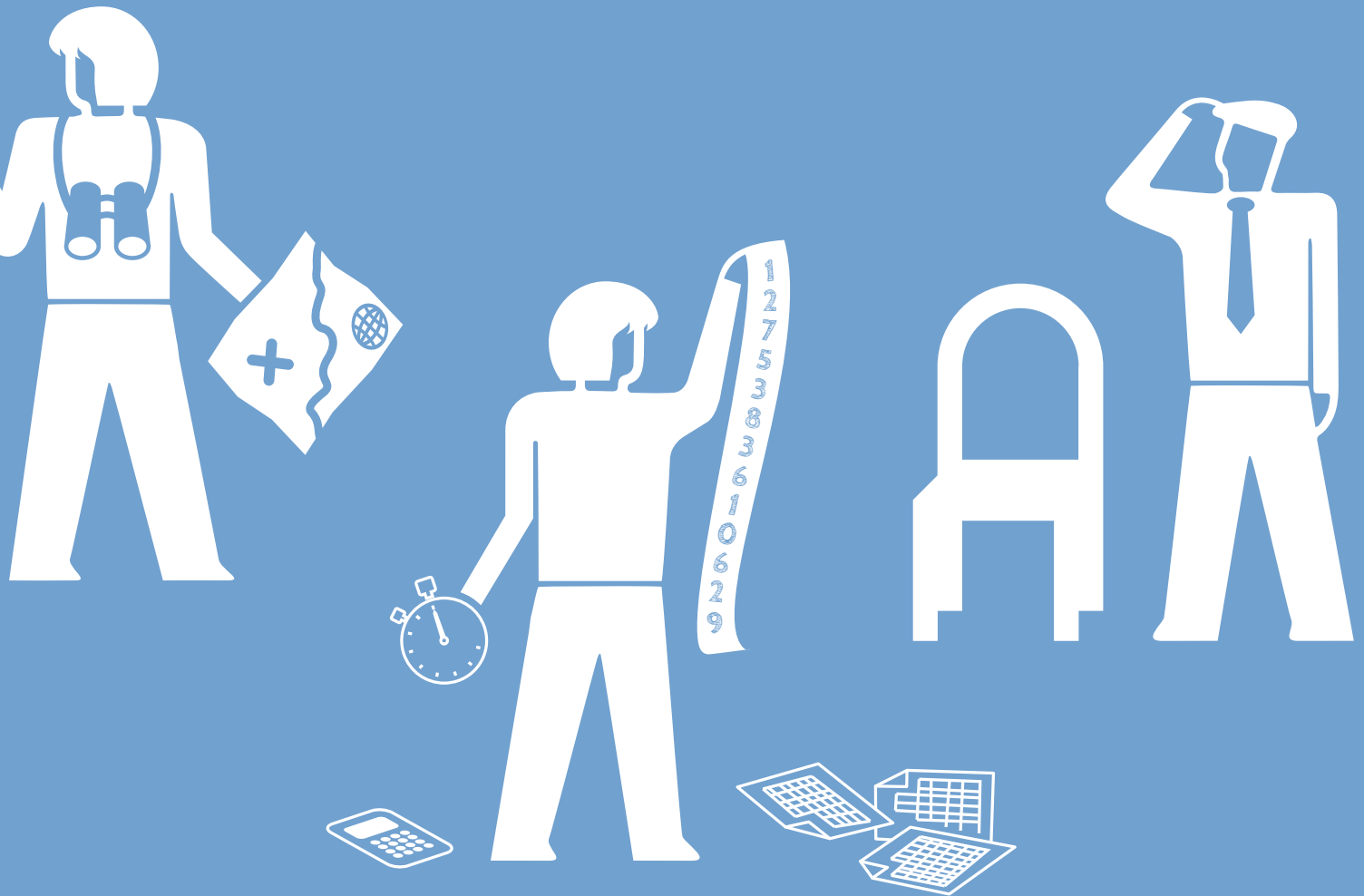
journeys

# Two ways to





# read this book



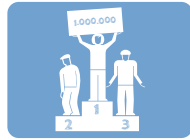
We didn't intend this book to be read from start to finish – but you are more than welcome to do so. Instead, we've written this book with busy managers in mind – people, like you, who don't have the time to read the whole book carefully but only the sections that are relevant to your particular business.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

*Open Source*

### 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.

*Open Source*

### 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.

*Servitization*

### 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.

*Agile*

### 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.

**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.





# 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

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

SCENARIO / Open Source

# 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.

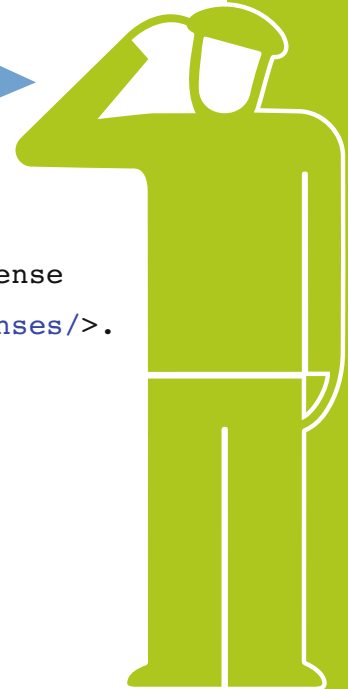
You should have received a copy of the GNU General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

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.

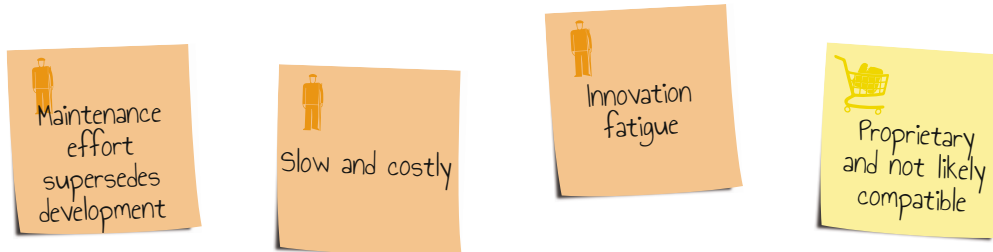


In recent years, this image of OSS developers has become outdated, and many companies are now actively participating in OSS communities for a variety of reasons.

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 where 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.



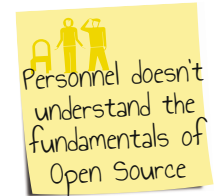
Even if the work with Open Source preferably ought to be sanctioned by management it often starts as skunkwork\*.

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.

So the engineers decide 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.



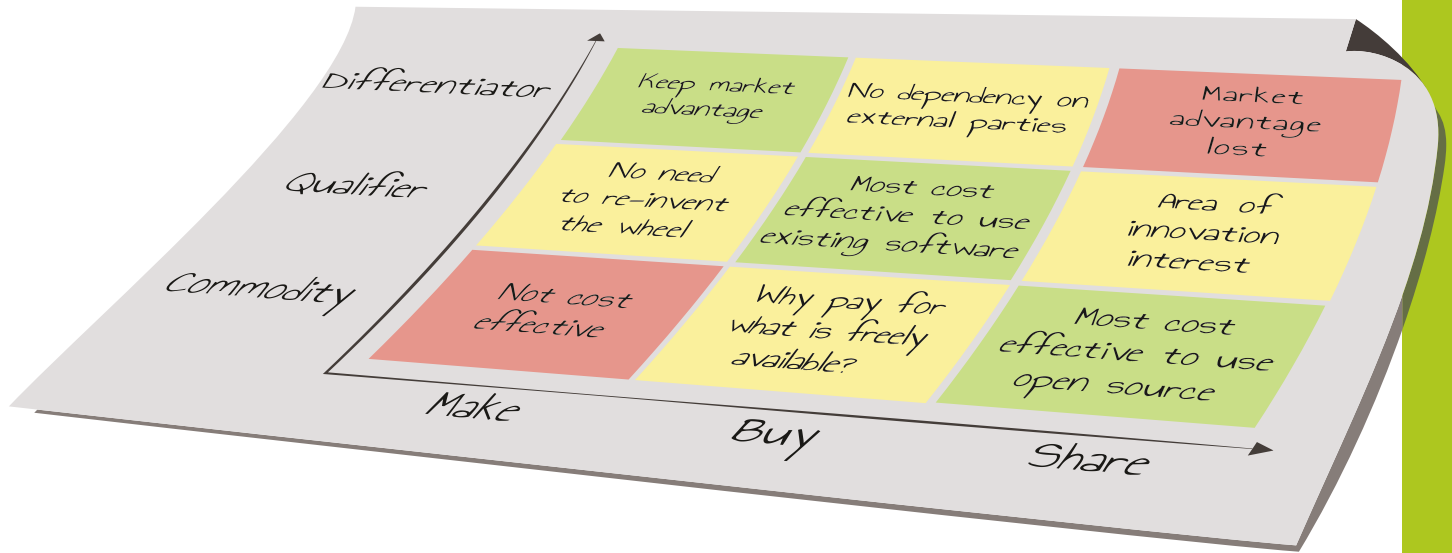
Initially, the likely main challenge for management is that Open Source is perceived to raise an unknown legal risk. Management may also perceive that with Open Source unknown technology, with unknown effects on the company's offering, would flow in uncontrolled, and worse, that valuable corporate Intellectual 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.



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 direct-ed 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 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.

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.

Volunteer contributors will spring up freely to contribute to interesting projects



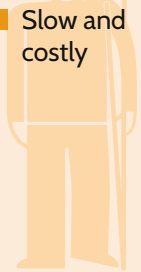
■ Accelerate speed of development

■ Share development costs

■ Improve innovation



- Innovation fatigue
- Maintenance effort supersedes development
- Slow and costly

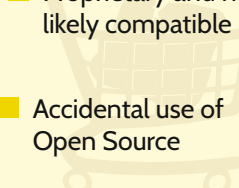


- Not-invented-here culture
- Personnel doesn't understand the fundamentals of Open Source
- Management fears Open Source (a hacker thing)
- Management doesn't understand the potential with Open Source

- Closed (internal) and non-collaborative
- Legal is not part of the process and fears Open Source
- No license compliance check



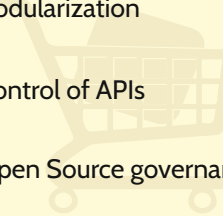
- Proprietary and not likely compatible
- Accidental use of Open Source



- Open Source Competence & Developer Program
- Open Source Officer - Cooperation 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

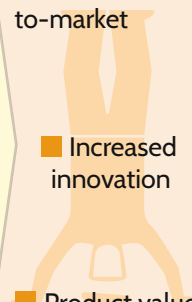


■ Reduced development and maintenance costs

■ 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.



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.

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.

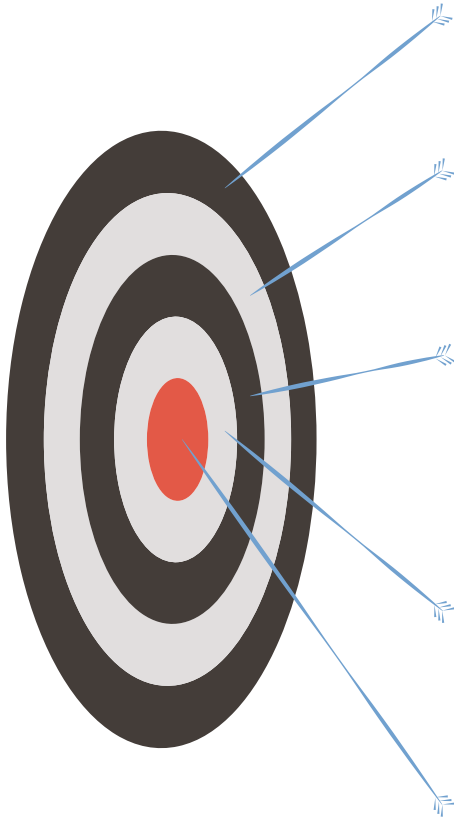
## Business Driven

### 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.



## 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.

■ Accelerate speed of development

■ Minimize maintenance cost

■ Share development costs

■ Increase innovation

■ Influence/Drive industry standards

■ Slow development

■ Maintenance effort supersedes development

■ High OPEX

- 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 may be OK as it is "free of charge"

- Open Source Competence & Developer Program
- Open Source Officer - Cooperation with Legal & IPR
- Open Source Community Culture
- Participatory culture

■ Control Intake, Compliance, and Contributions

- Industry-wide collaborations
- Make-Buy-Share analysis is part of intake

■ Modularization

■ Open Source governance tools

■ A mix of strategic and proprietary components and a huge amount of openly shared components

■ Reduced development and maintenance costs

■ Increased innovation

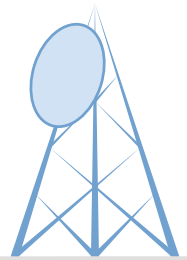
■ Reduced time-to-market

■ Faster growth than competition



CASE STUDY / Co-develop in a community

# 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

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



CASE STUDY / Co-develop in a community

# 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



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.

■ Accelerate growth of business

■ Shorten time to market

■ Decrease development cost per product



■ Limited or no competence in Open Source

■ Classic development organization

■ Fairly new in software development



■ Not sufficient understanding of risks and benefits with Open Source

■ Develops everything self



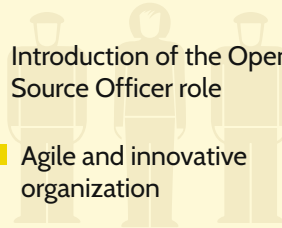
■ Limited possibilities to join or drive industry standards

■ Proprietary products



■ Introduction of the Open Source Officer role

■ Agile and innovative organization



■ Intake process in place

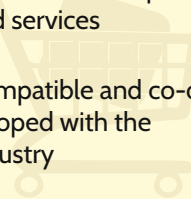
■ Compliance and license processes in place

■ Contribution and community policy in place

■ Use of a Make-Buy-Share strategy

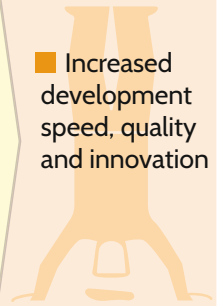
■ Innovative, new products and services

■ Compatible and co-developed with the industry



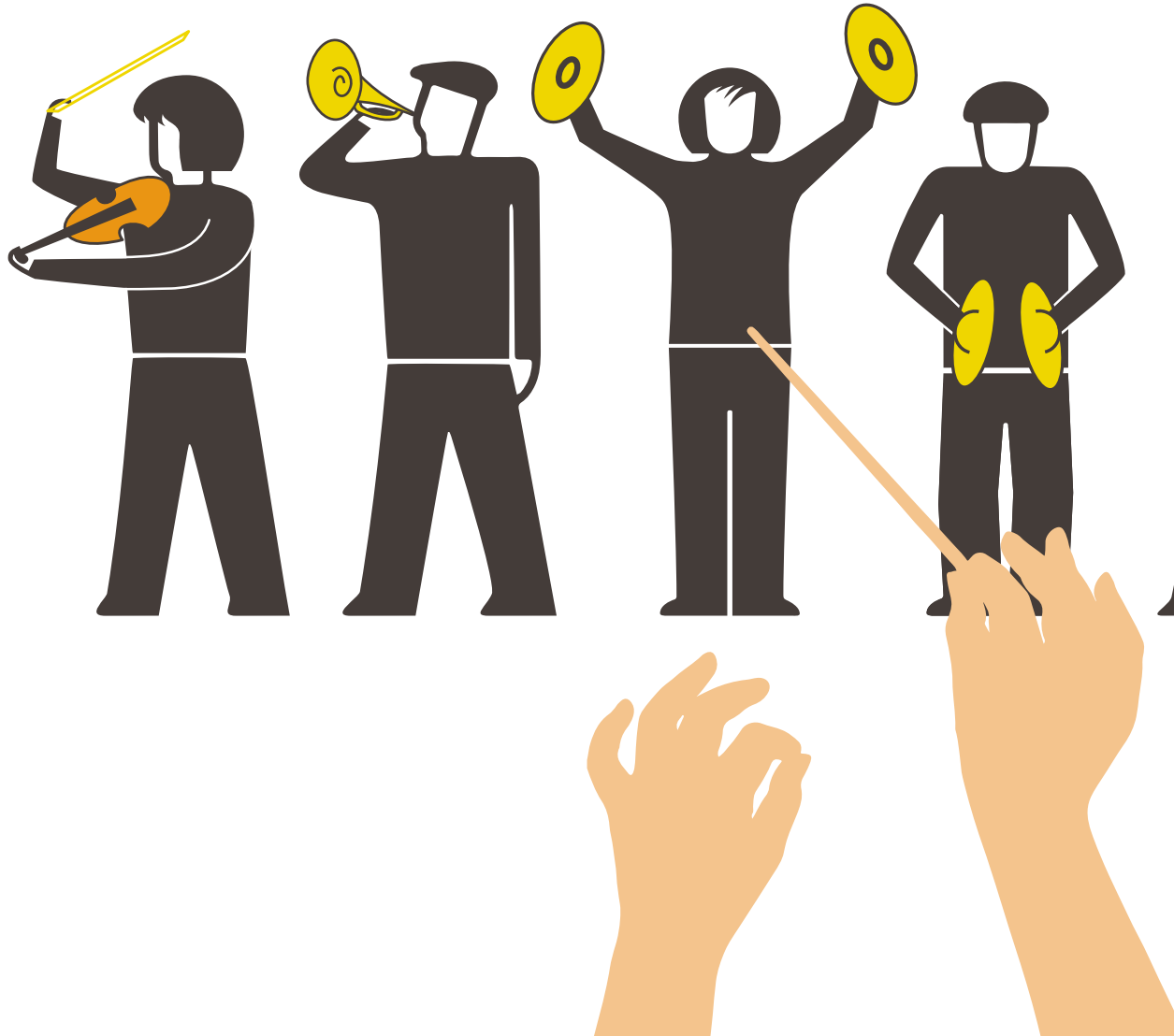
■ Full understanding of the risks and benefits of Open Source on an engineering and business level

■ Increased development speed, quality and innovation



■ Ability to co-develop and drive or lead the standards in the industry

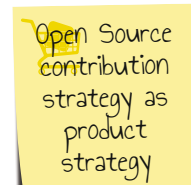
# Building ecosystems





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.

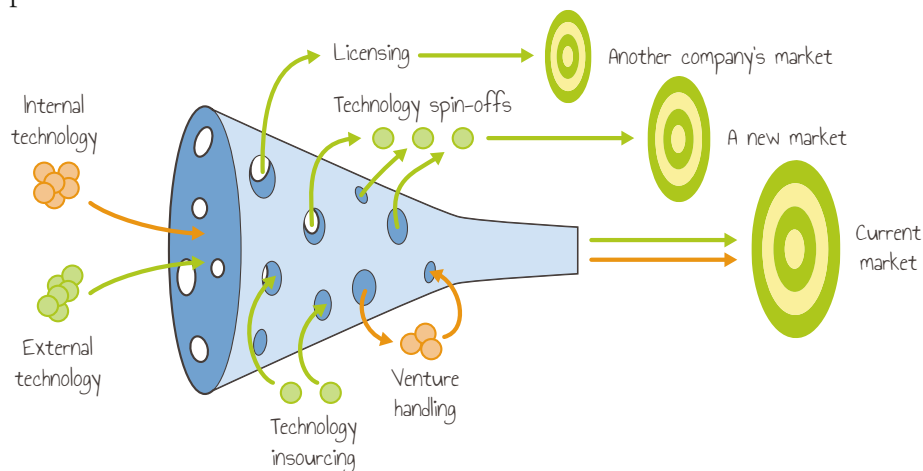
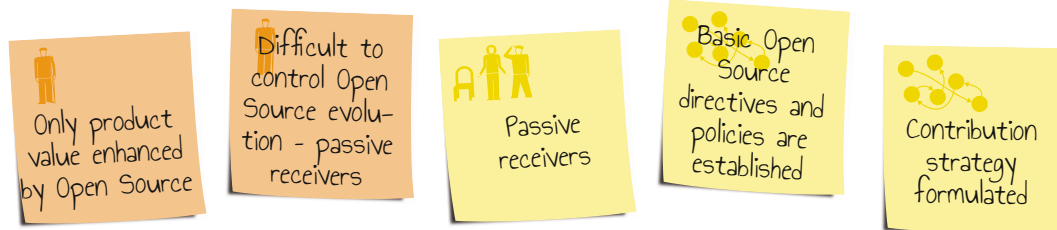
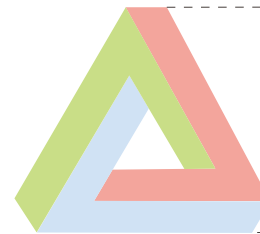


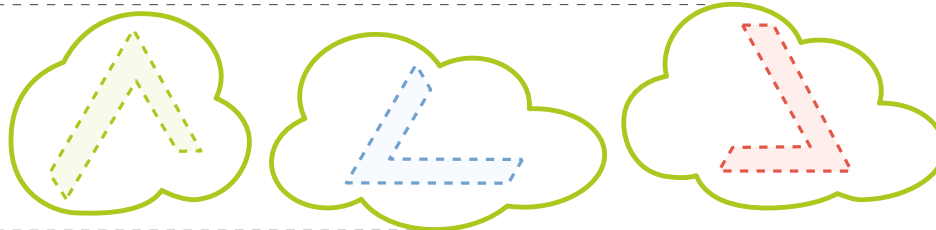
Illustration based on picture from presentation of Open Business Models by Henry Chesbrough

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.

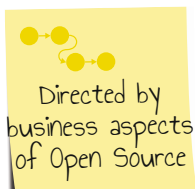


However, a journey like this can only start once a company's use of Open Source can be said is being directed, when the company has gained such support from the executive management so the use of Open Source software is paramount of the product management. (See the chapter **Co-operate in a community**.) 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.





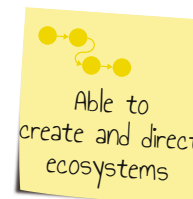
Although the entire idea with building an ecosystem is to scale the business, it can still be a challenge for the management to recognize how Open Source as a phenomenon can be used as a business tool. The management will have to explore radically different market logic, and in this, they have to fundamentally 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.

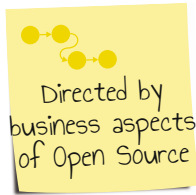


Most essentially comes with Open Source a participatory culture. Customers are moving from being passive receivers of purchased solutions to actively involve themselves in the development of the product offering itself. Many companies have created Developer Programs to catch and nurture this kind of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





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

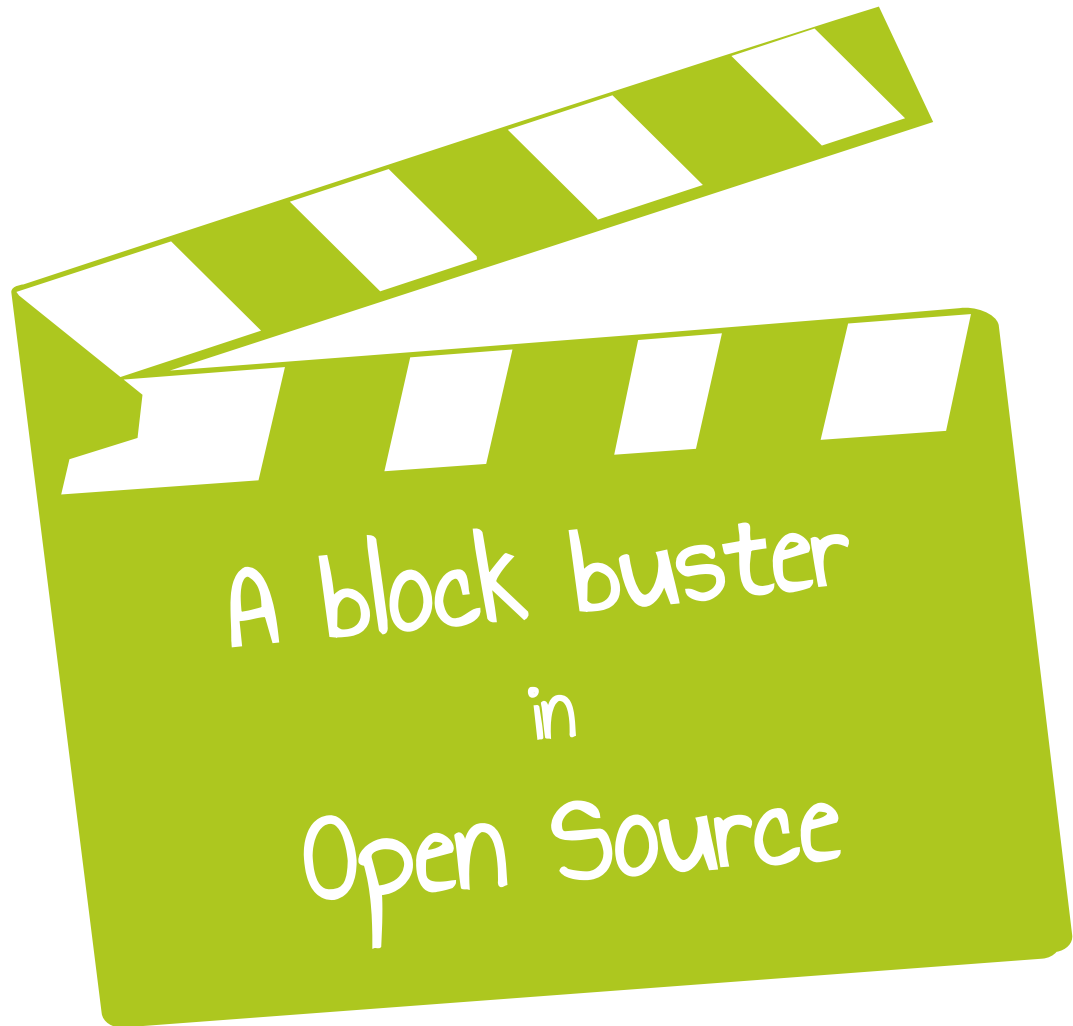
Open source and ecosystems are almost always the preferred way when scaling a business from products into services. Take again Netflix, who provided a DVD-by-mail service before creating their video streaming service. They likely made a transformation such as the one the Servitization scenario describes.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





CASE STUDY / Building ecosystems

# 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

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](http://www.businessweek.com)

Netflix Announces \$100,000 in Prizes for Coders, 2013, [www.businessweek.com](http://www.businessweek.com)

Netflix and YouTube Dominate Online Video. Can Amazon Catch Up?, 2013, [www.businessweek.com](http://www.businessweek.com)

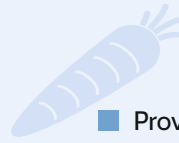
NetflixOSS Meetup, 2013, [www.slideshare.net](http://www.slideshare.net)

[techblog.netflix.com](http://techblog.netflix.com)

## Netflix Open Source software downloads

[netflix.github.io/#repo](https://netflix.github.io/#repo)

■ Accelerate growth of business



■ Secure leadership in streaming video technology

■ Provide the best UX for streaming video

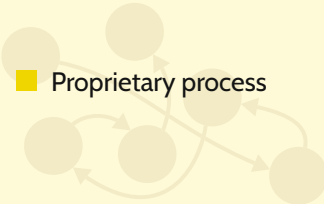
■ Slow development

■ Internal focused development



■ Difficulties of securing high demands on robustness

■ Proprietary process



■ Proprietary technology



- Highly paid top talent developers
- Developers want to be part of an attractive community
- Participatory culture
- Highly skilled in cloud computing
- Authority on open source

■ Hired developers that share almost all code freely

■ Industry-wide collaborations

■ Leader of the evolution of streaming technology

■ Strategic technology opened up as a market disruptor

■ Open source contribution strategy as product strategy

■ Attractive and interesting product

■ Robust and high quality product

■ Increase sales and profitability

■ Value and new business extracted from own ecosystem

■ Faster growth than the competition

■ Attracting the best developers

# 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:

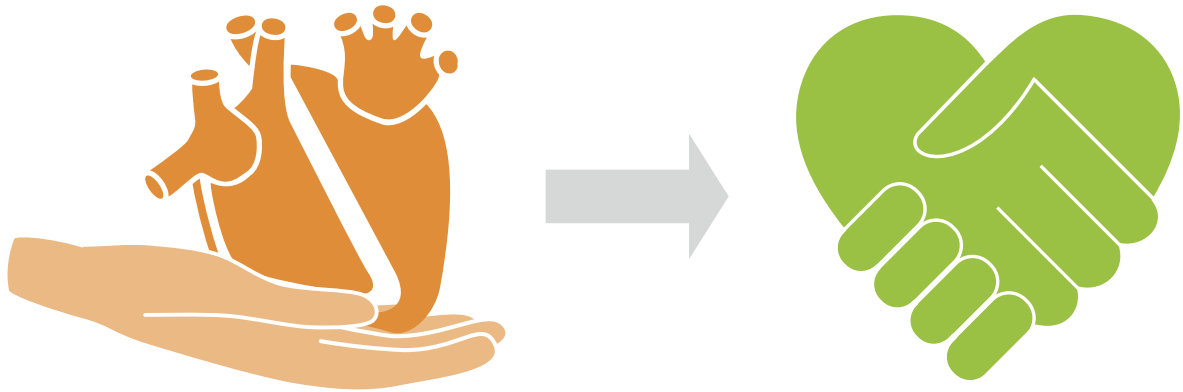
- **Made to Serve:** How Manufacturers can Compete Through Servitization and product Service Systems, Timothy Baines
- **The Startup Owner's Manual:** The Step-By-Step Guide for Building a Great Company, Steve Blank

\* ITIL 2011, (Information Technology Structure Library)



# 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.

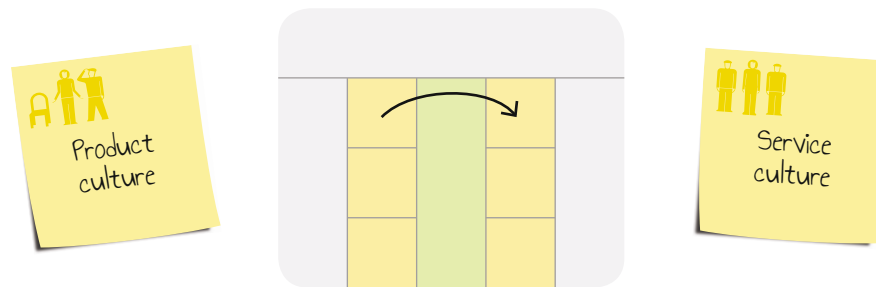


Least complex is to keep the product and purchase-based business model, but add supplementary services such as maintenance and premium programs.

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.

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.



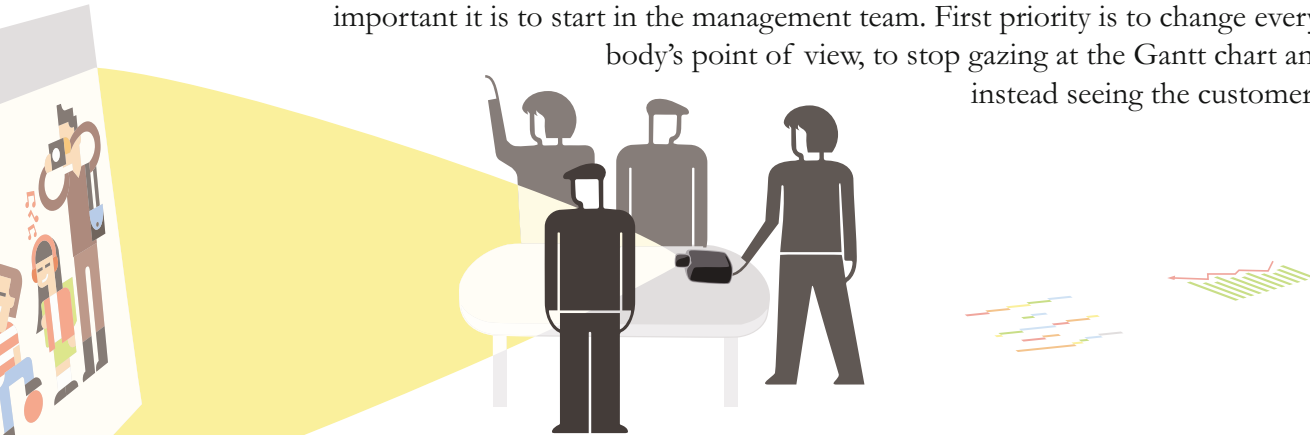
Consider for a minute the dynamics within the management group. Who gets to talk most on the management group meetings? This will change. Substantial amount of time and resources that are currently allocated to track and report development and manufacturing will decrease to an absolute minimum. Sales, 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.

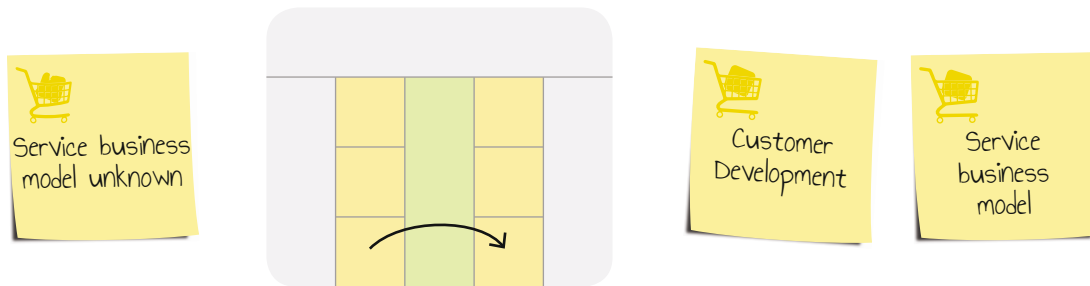




Regardless where you aim in your scaling effort, it cannot be emphasized enough how important it is to start in the management team. First priority is to change everybody's point of view, to stop gazing at the Gantt chart and instead seeing the customers.



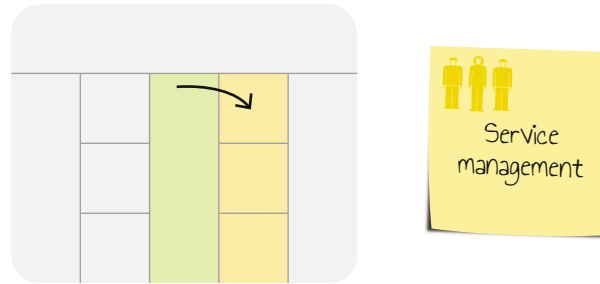
Another priority, and indeed a great challenge, is to develop a scalable, repeatable and profitable business model. In terms of creating it, there isn't really any difference between an established product business that wants to start a service and a start-up business. They have an idea of a service 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

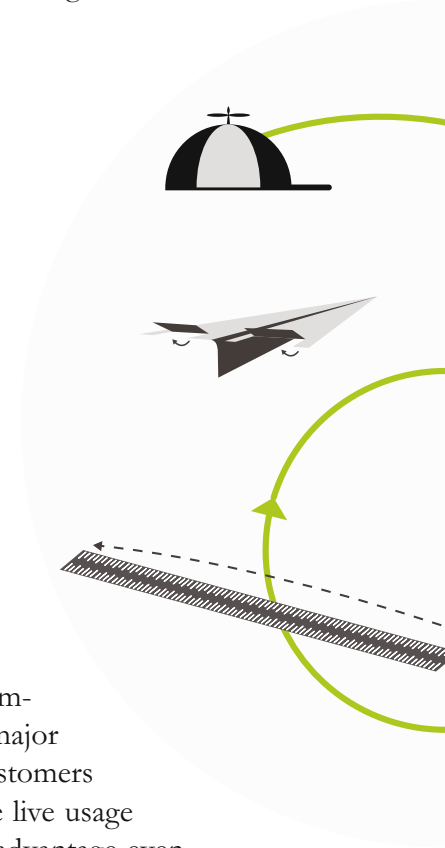
Customers have to get value day after day, month after month, to stay as such. They have very high demands on availability. When at least 99.9% availability is expected on a normal service, there's absolutely no room for temporary glitches. The ITIL 2011 has shown to be a great source of knowledge to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

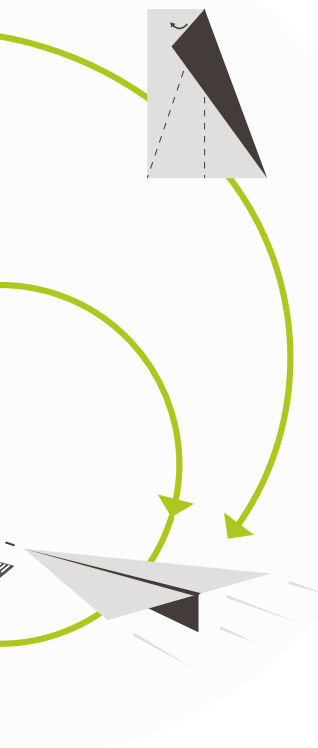
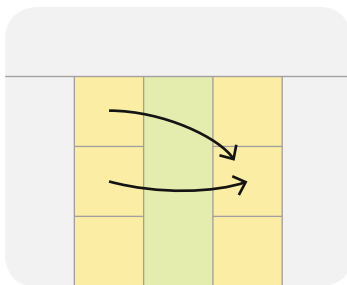
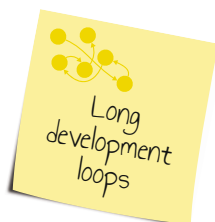


How do you provide a service that customers willingly pay for month after month? You have to get to know them and demonstrate that your service adds value to every one of them, every day. The technology gets secondary – the customer experience becomes primary. What are their needs? Understanding your 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.

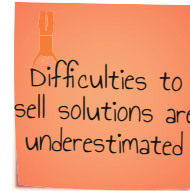
After having provided the new features to your customers, you can measure, analyze and improve them. If a feature doesn't live up to the customer's expectations, the design has to be improved. New features are not taken away from the developer's backlog until the customers show, either through surveys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



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:

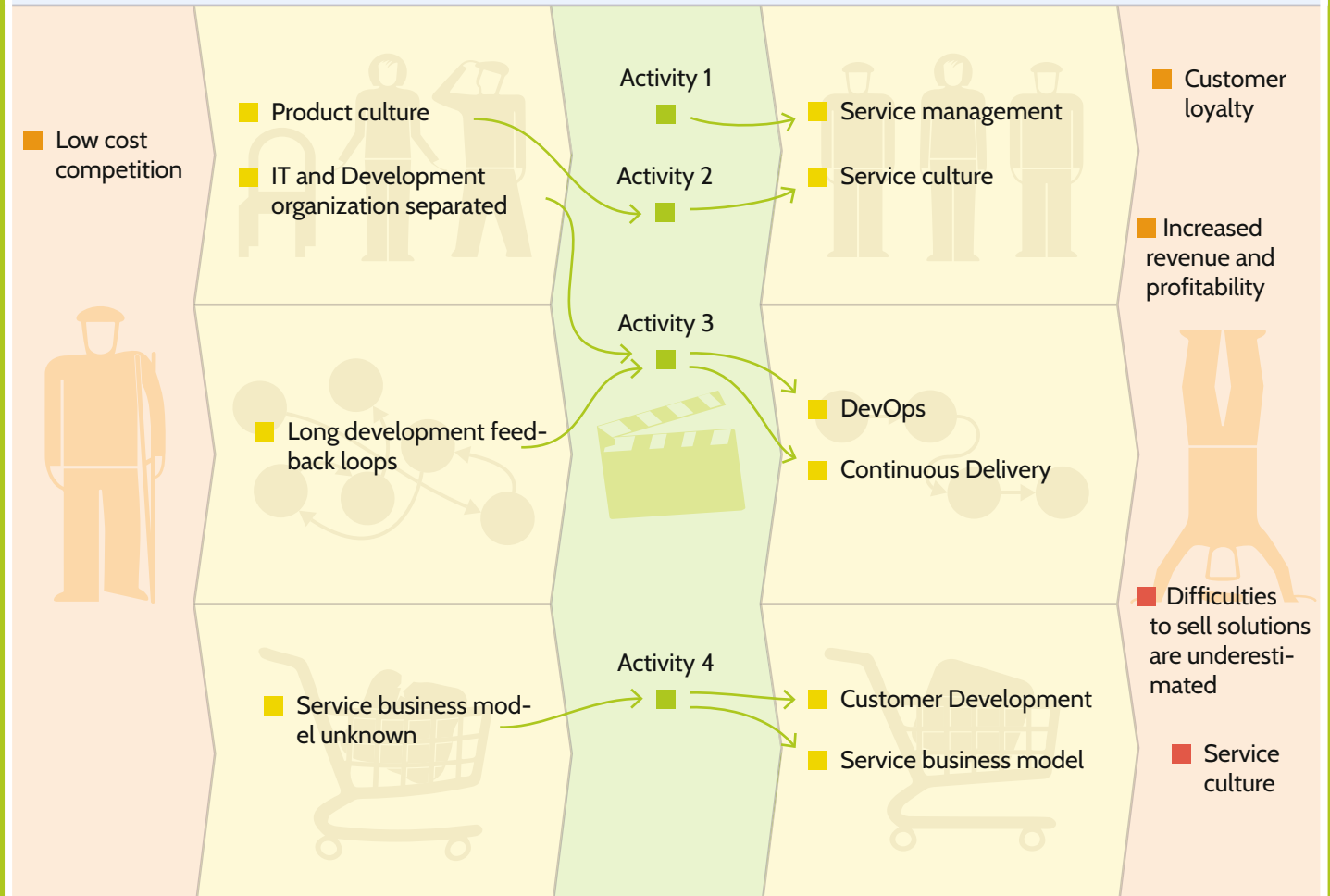
- Customer loyalty
- Increased revenue and profitability
- Robust cash flow and revenue streams

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

■ Customer expectations

■ Financial incentives

■ Gaining competitive advantage



## 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.



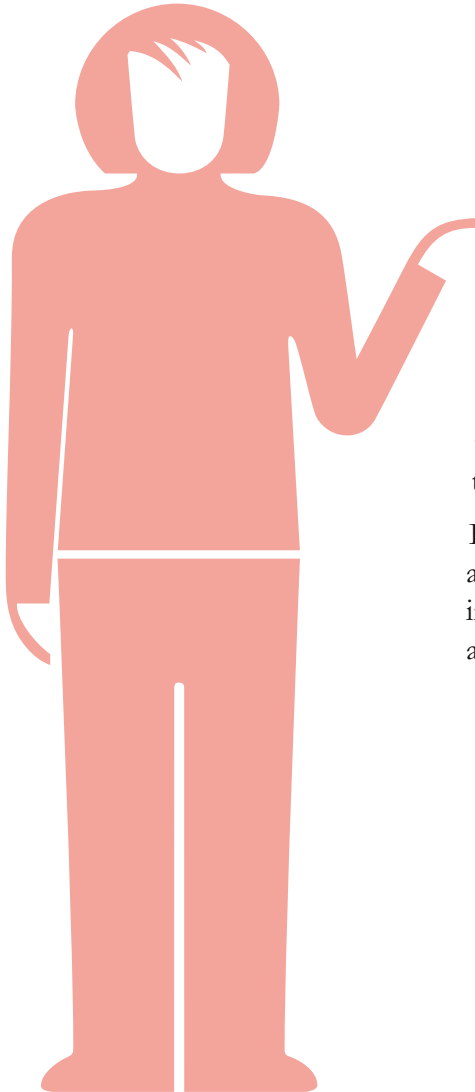
### **Adding Internet to things**

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.

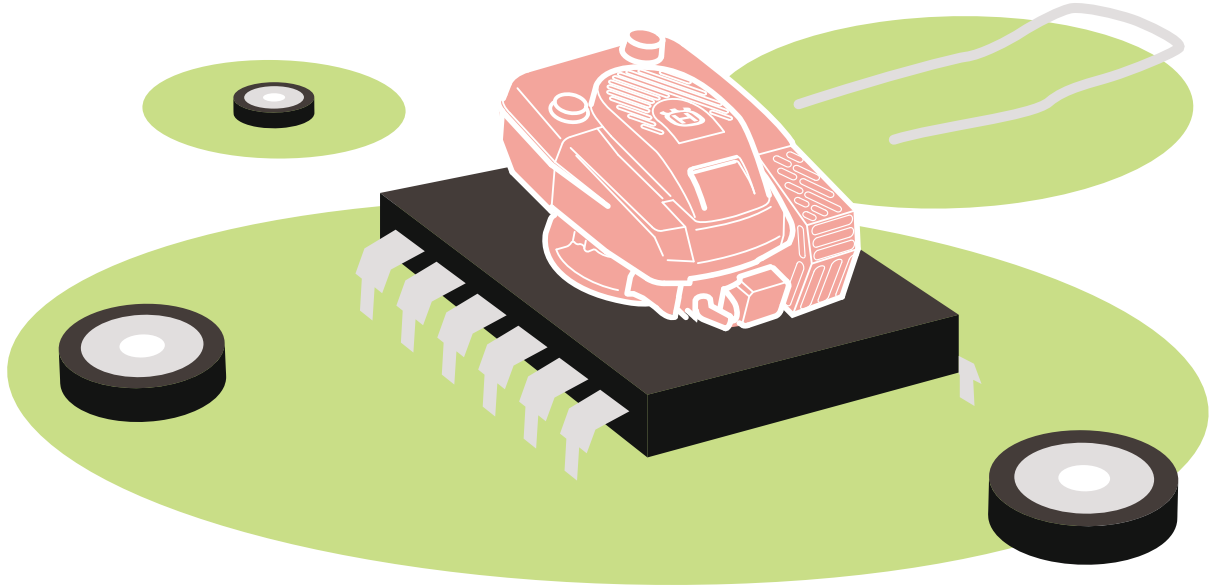
The chapter **First things first** describes a canvas that was created for this purpose, as a guide on how to improve the software's internal structure.

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**.



CASE STUDY / Add supplementary services

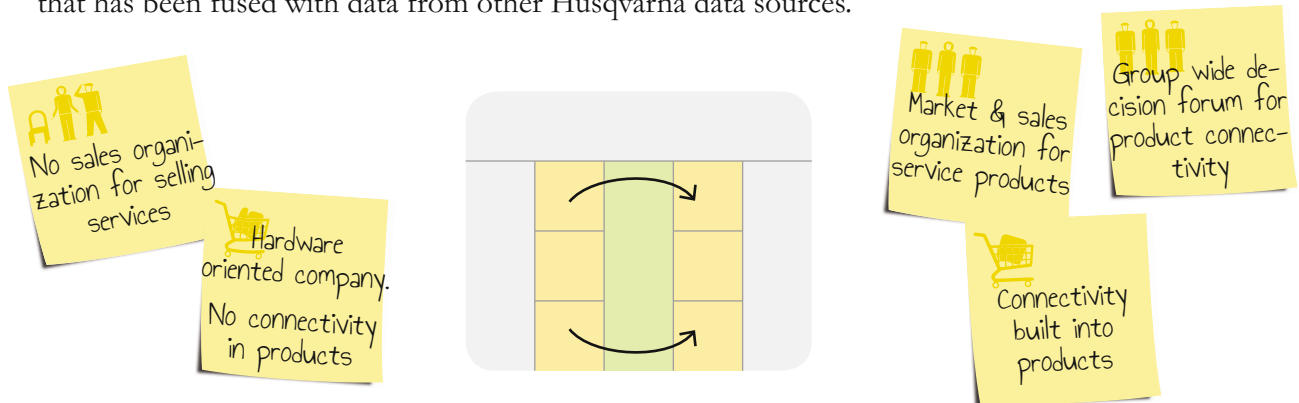
# Adding Internet to things



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.

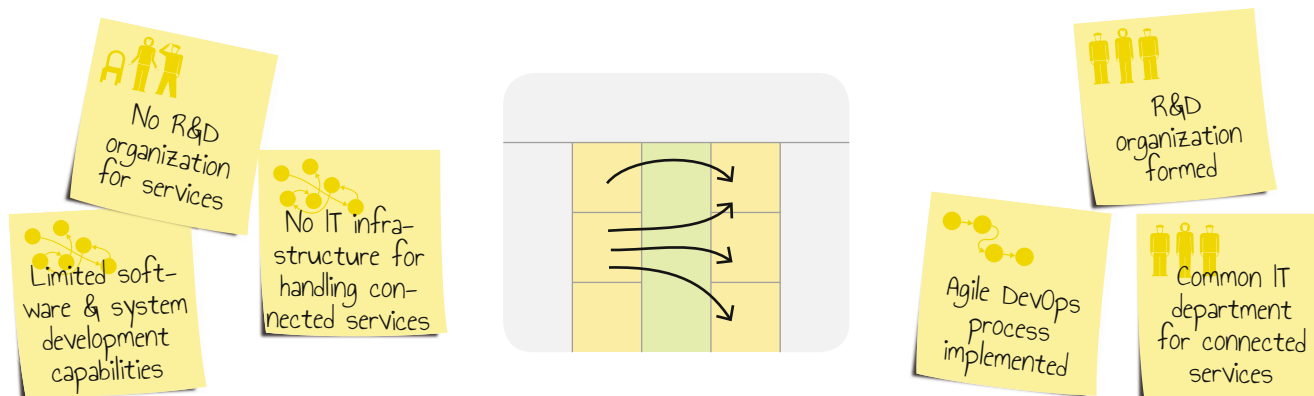


Husqvarna Fleet Services started in 2007 as an idea in the After-sales department, basically to offer a service that help Commercial Lawn and Garden customers managing their fleet of Husqvarna products. Until 2011, all development was run by After-sales as a study on a very limited budget. To create 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.

Customer expectations

Gaining competitive advantage

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.

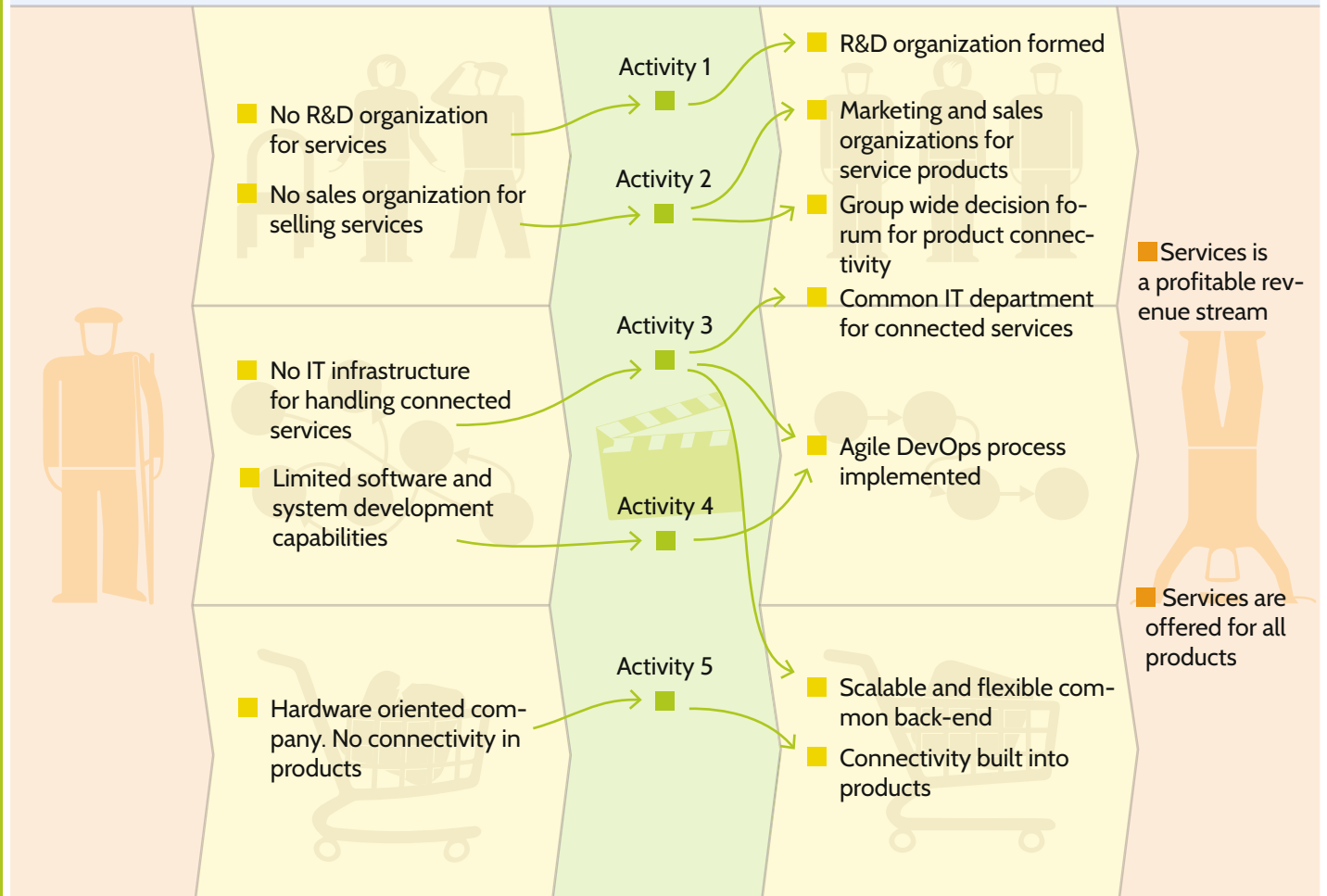


Husqvarna saw the demand from the market, the need for a product like Husqvarna Fleet Service. They identified a market trend such as digitalization and Internet of Things and now the organization is changing accordingly. What originally started as an experiment is now reshaping the identity of the entire 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.

■ Customer expectations



■ Gaining competitive advantage





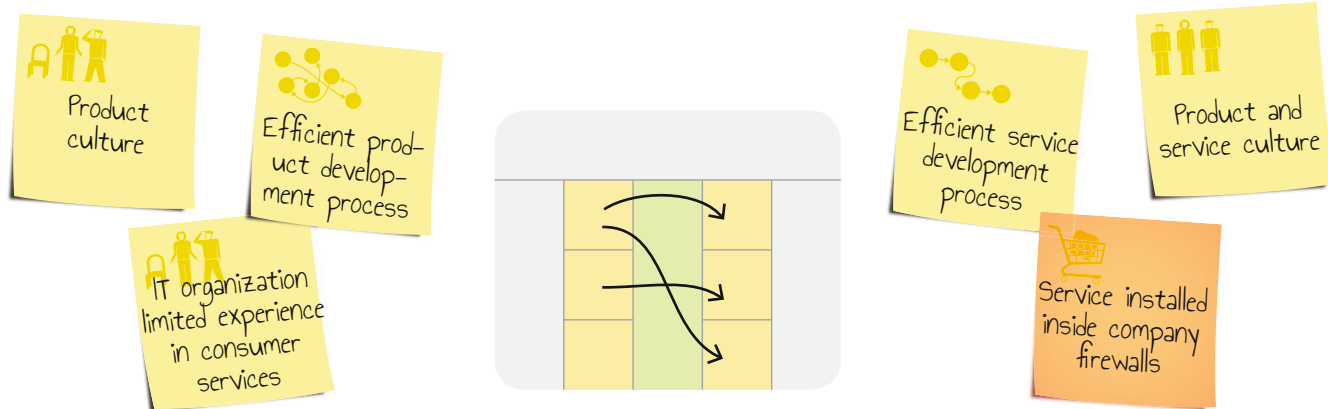
CASE STUDY / Add supplementary services

# 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.

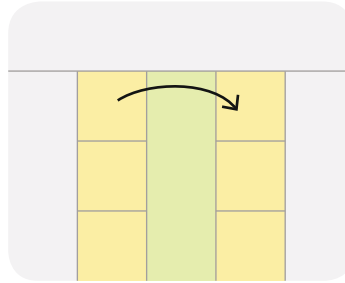


Sony Ericsson was a product company with very little experience in delivering services. The service organization experienced huge difficulties in both changing the culture of Sony Ericsson as educating personnel in what it meant to deliver a service. Part of the explanation is that almost all sales came 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](http://www.consultancy.nl)

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.

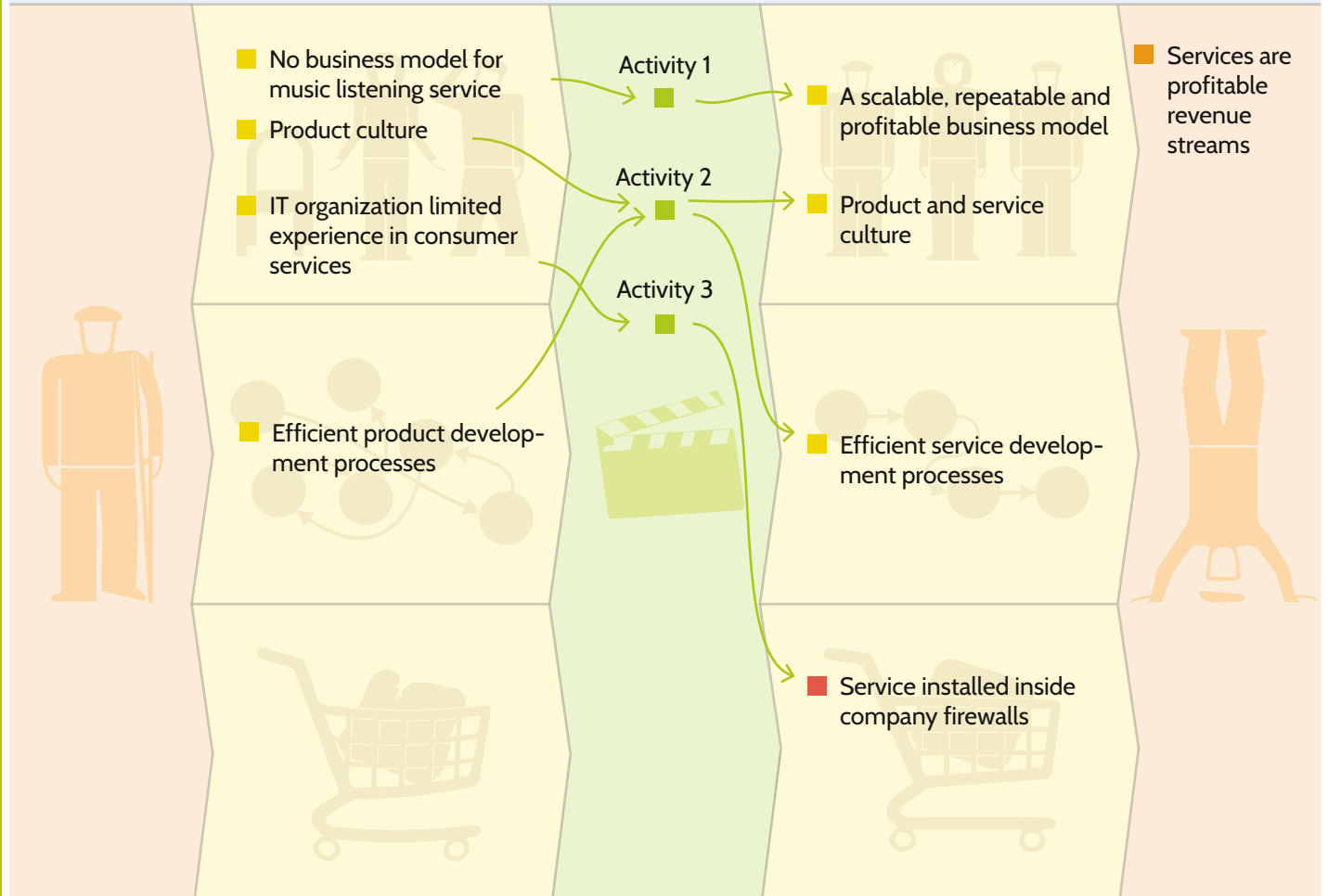
Spotify reported a loss of \$4.4 million in 2008. But, as it had a long-term goal to be profitable, it could take the loss for an extended amount of time. PlayNow plus was never considered to be a future profit maker, nor was it seen as an investment. The service was terminated in 2010, as a decision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



■ Increase sales of mobile phones



■ Leading the global music listening business

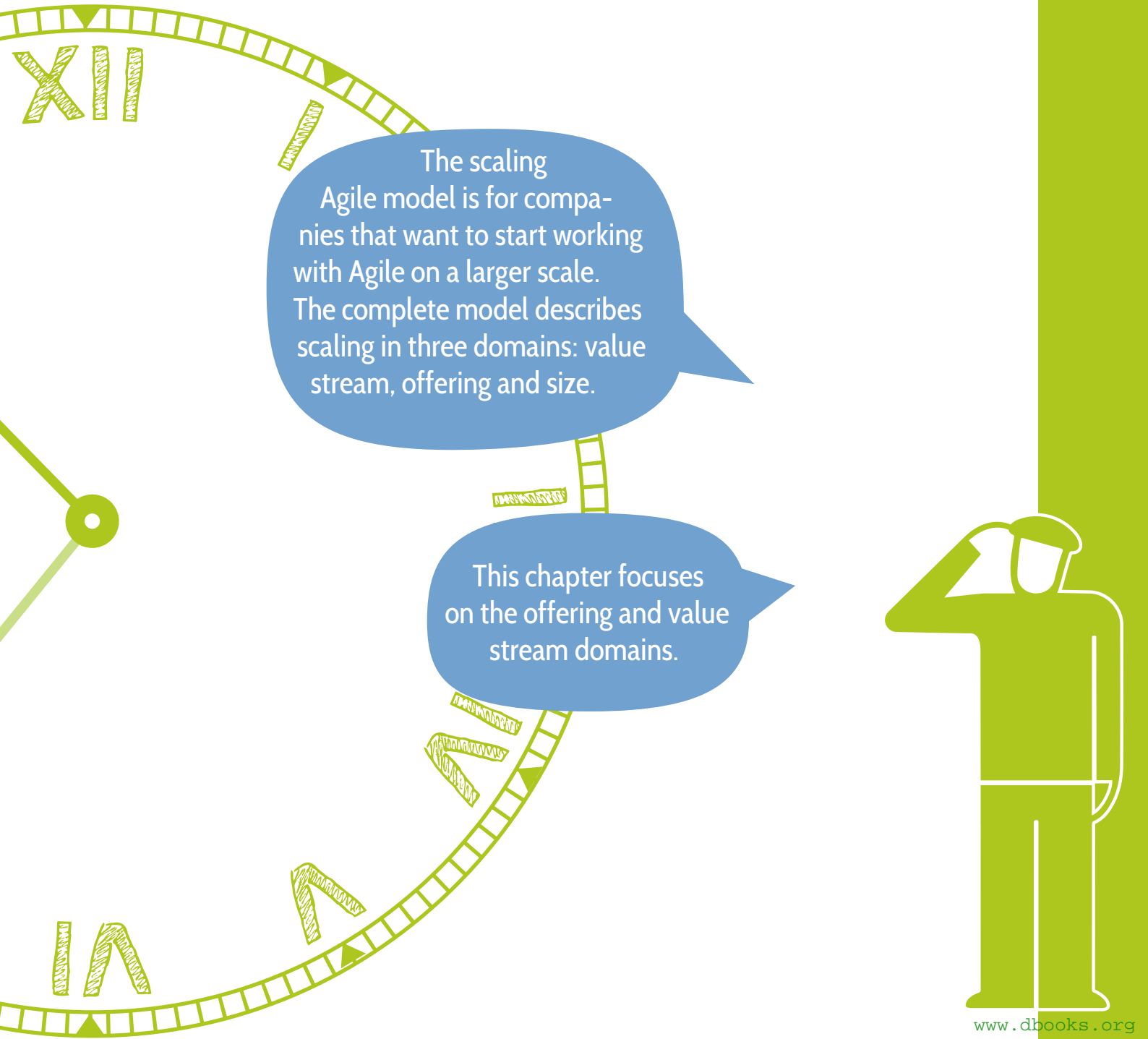


SCENARIO / Agile

# Deliver 24/7







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: value stream, offering and size.

This chapter focuses on the offering and value stream domains.

## 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:

1

### 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.

2

### 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.

3

### 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.

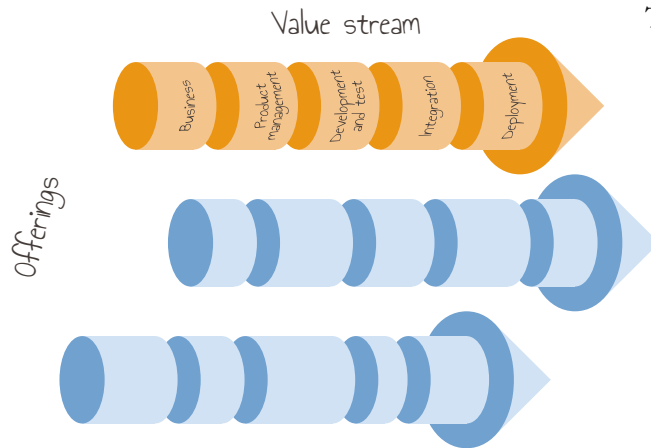
## Scaling agile

Customer value

Differentiation

Innovation

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.



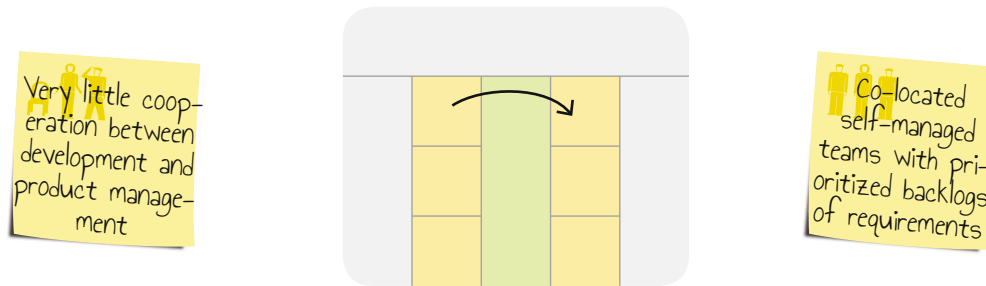
There are three reasons why a company would want to scale up agile to the whole organization:

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

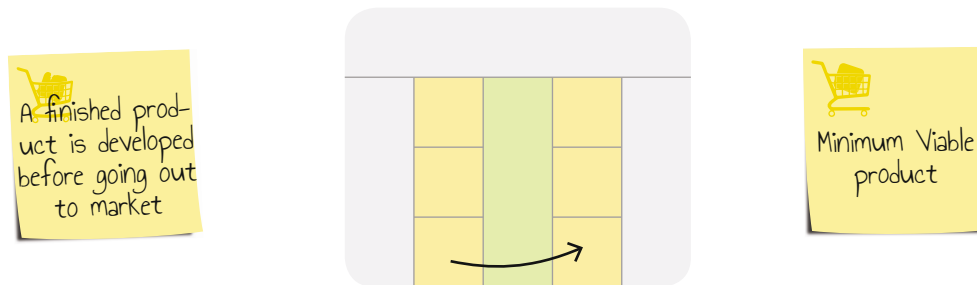
Everything is sprung out of the need to become stronger, to innovate better products and to better differentiate between products.

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 it takes to keep the customer ahead and the competition behind.

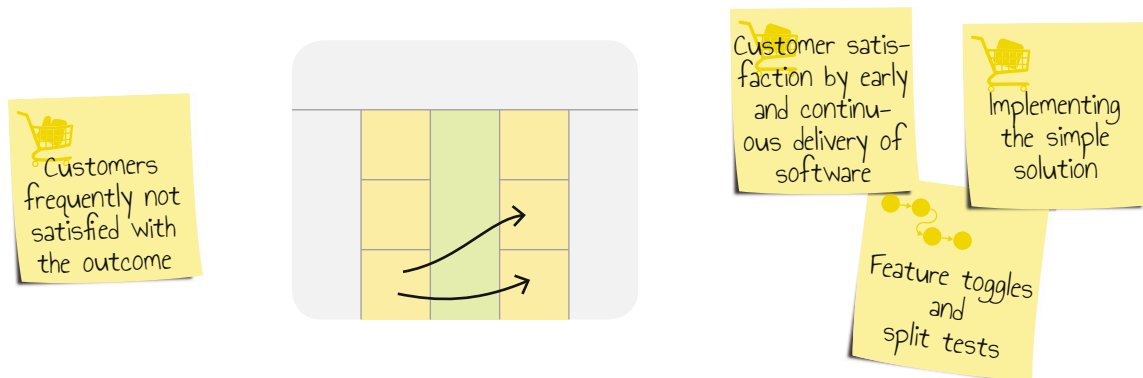


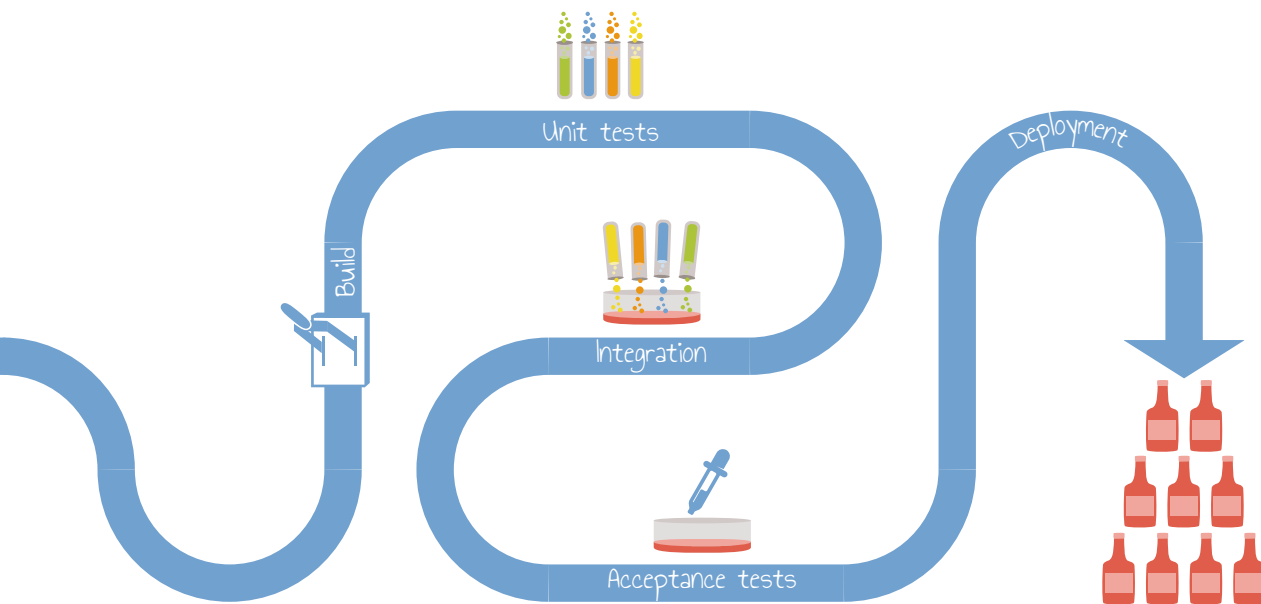
Once a company has established an agile value stream, it becomes easier to differentiate between various products and services. Of course, it's not as simple as just replicating teams, or worse, assigning the same team for several product offerings. Scaling in the product domain, creating new products 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

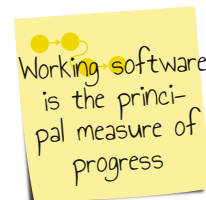
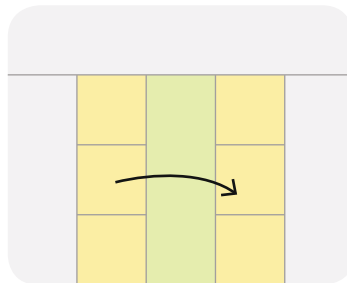
As innovation and coding can be dealt with by introducing agile methods, so can test, integration, build, and delivery practices. How about eliminating all manual work, developing an ability to deliver a change to the customer within hours, or even minutes? How about being able to deliver bug fixes soon 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.



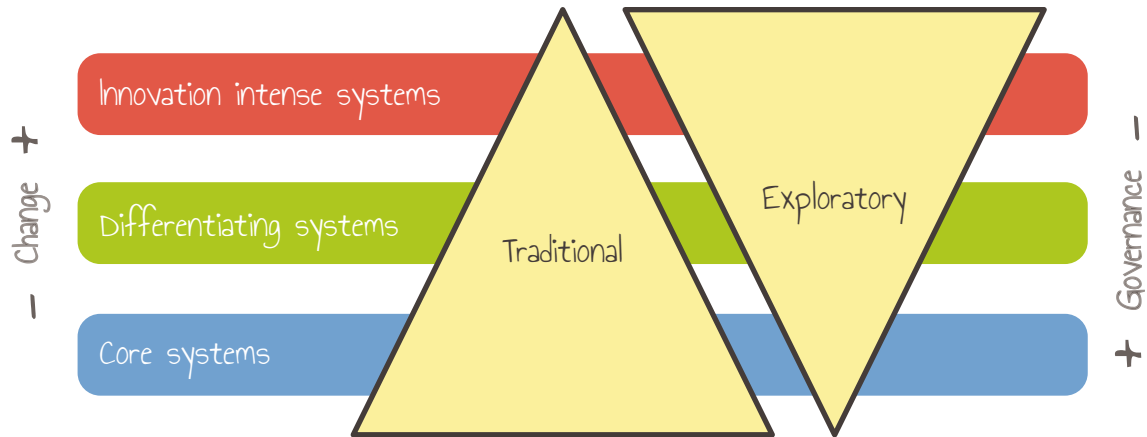


Continuous Delivery depends on a delivery pipeline – a series of steps that a product’s source code has to go through to be delivered. To establish such a pipeline isn’t trivial and doesn’t happen over a night. Apart from the very challenging task to automate delivery to customers, the biggest challenge 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.



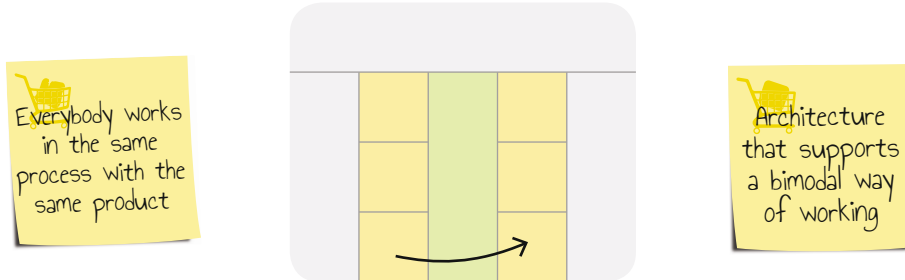
## Maintaining stability

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 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.

## 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.

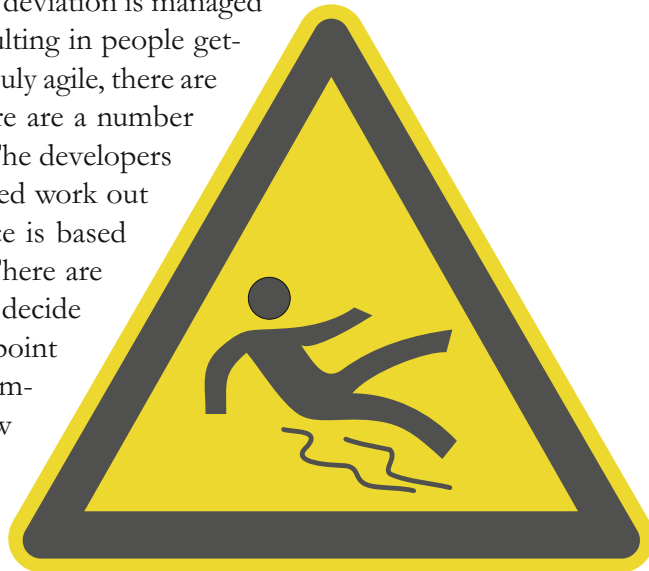
**Low maturity in a company's continuous integration and test procedures causes long quality feedback cycles.** To have successfully implemented procedures include at least daily integration and build, and some level of automatic testing. If the environment and processes don't allow for this, then further 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.



■ 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.