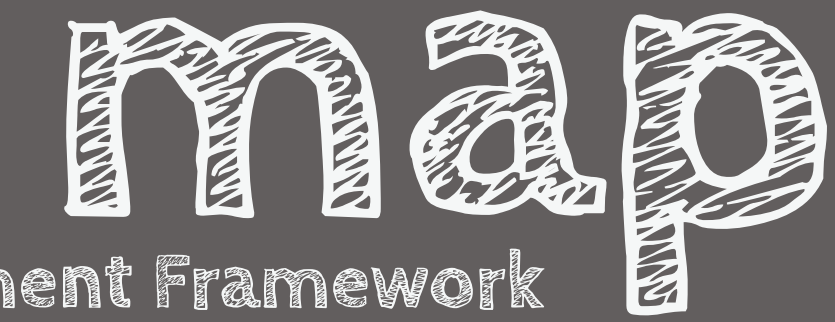
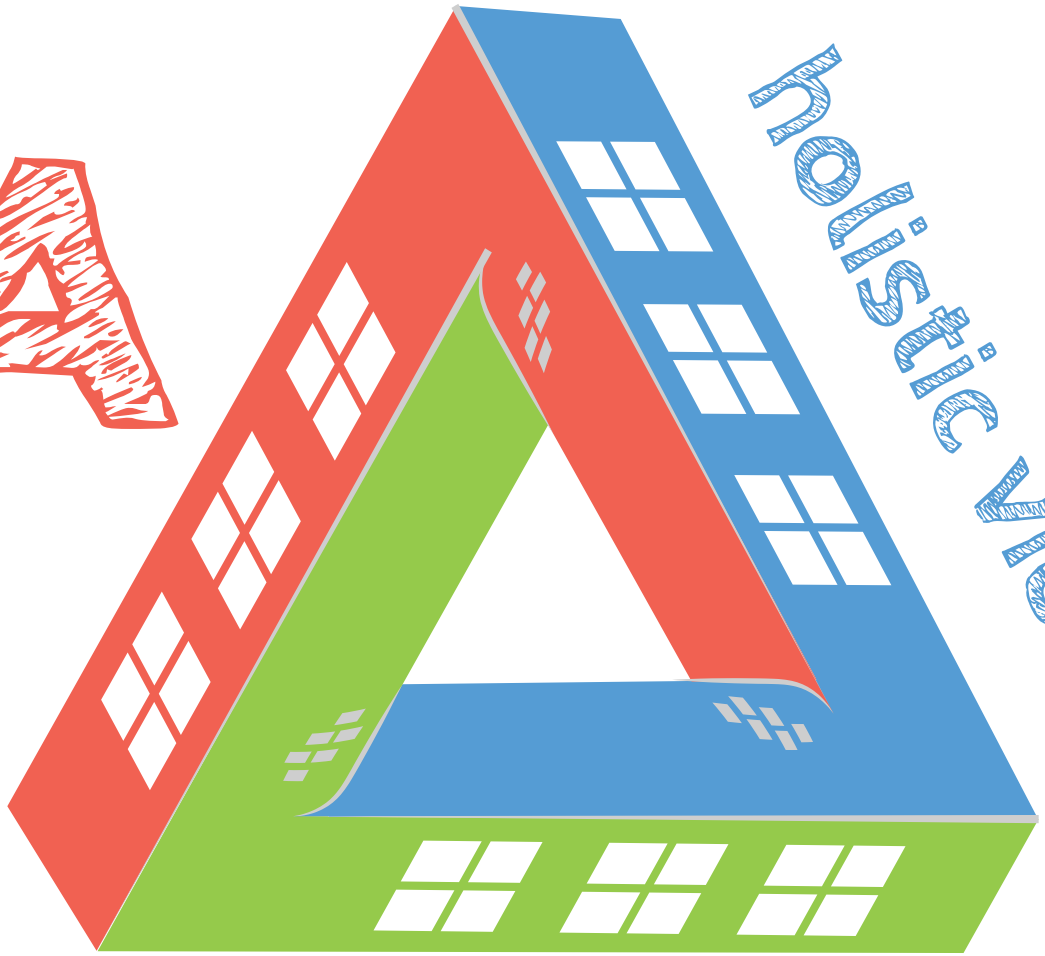
The

Scaling Managem

B. Fitzgerald et al., *Scaling a Software Business*,
DOI 10.1007/978-3-319-53116-8_1


map
nent Framework

A



holistic view

to change

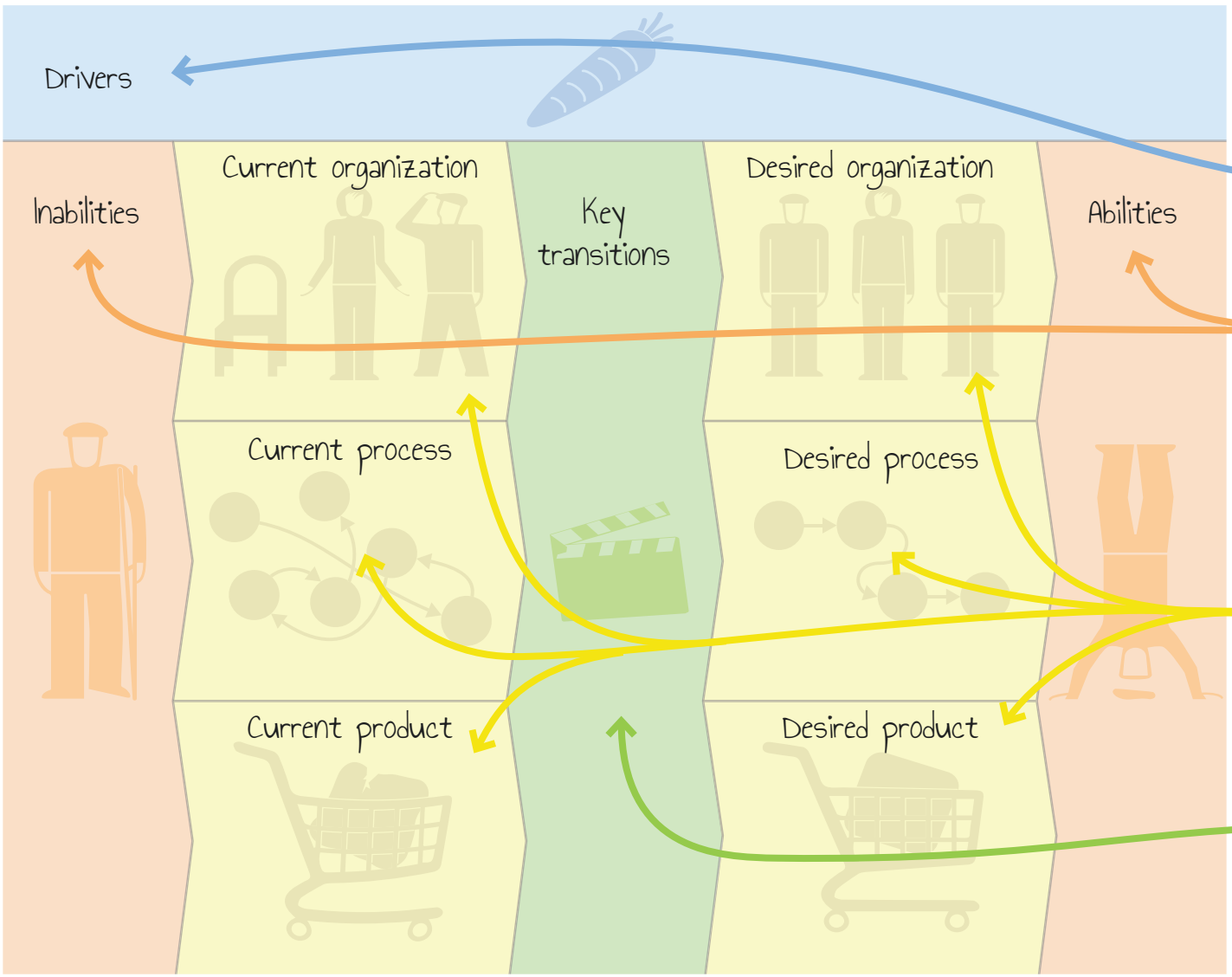
Scaling software development is a complex enterprise that can be organized in a number of ways. Since the early days of computing, hundreds, if not thousands of software development methods have been proposed. What is becoming increasingly clear, however, is that the software development function affects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

The Scaling Management Framework covers transformations in three domains

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

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

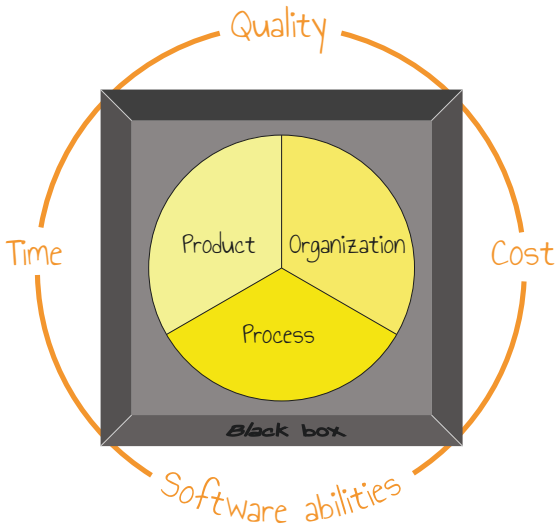
The framework is simple yet complete enough to suit all sorts of companies: small as large, local as global. It works equally well for companies that mainly deal with hardware but need to introduce software, as for companies that develop proprietary software and want to engage with Open Source communities.


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

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

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

>> Drivers >>

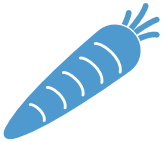


Next we observe the software development as a black box, focusing on performance, quality and other aspects that can be measured without knowing how they work. We know what we spend and how much we generate from the their work. These are basically the complaints and wishes of our top management, matters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

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

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

Drivers



The drivers of the need to change.

Inabilities

The inabilities that make the transformation challenging.

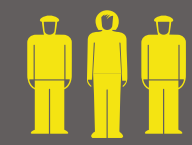
Desired abilities

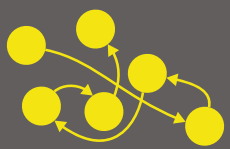
The measurable definition of done.

Current set-up

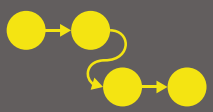
Desired set-up

Organization domain





Process domain



Product domain

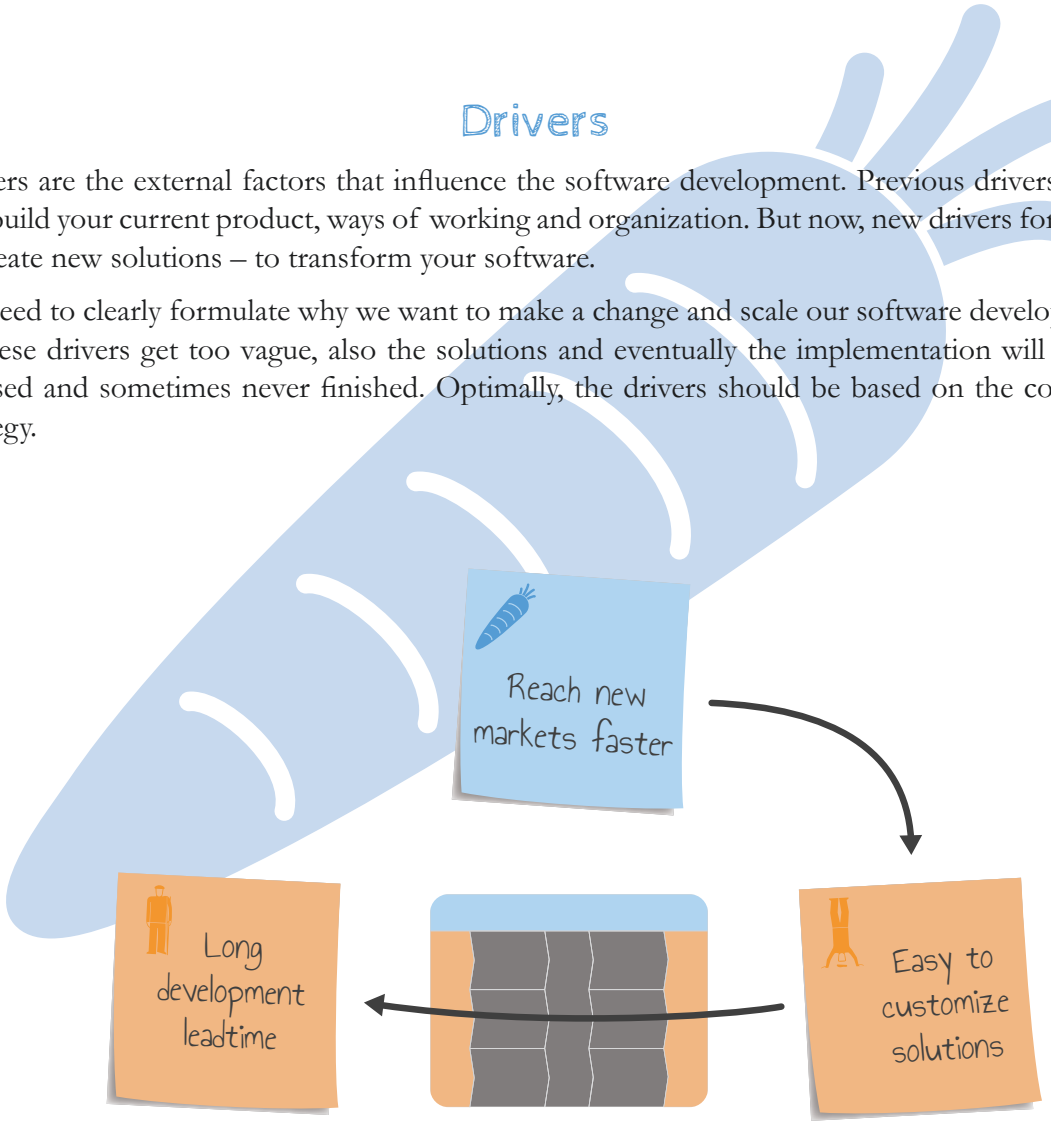
Transition

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

Drivers

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

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



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

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

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

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

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


1.000.000
1
2
3
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:

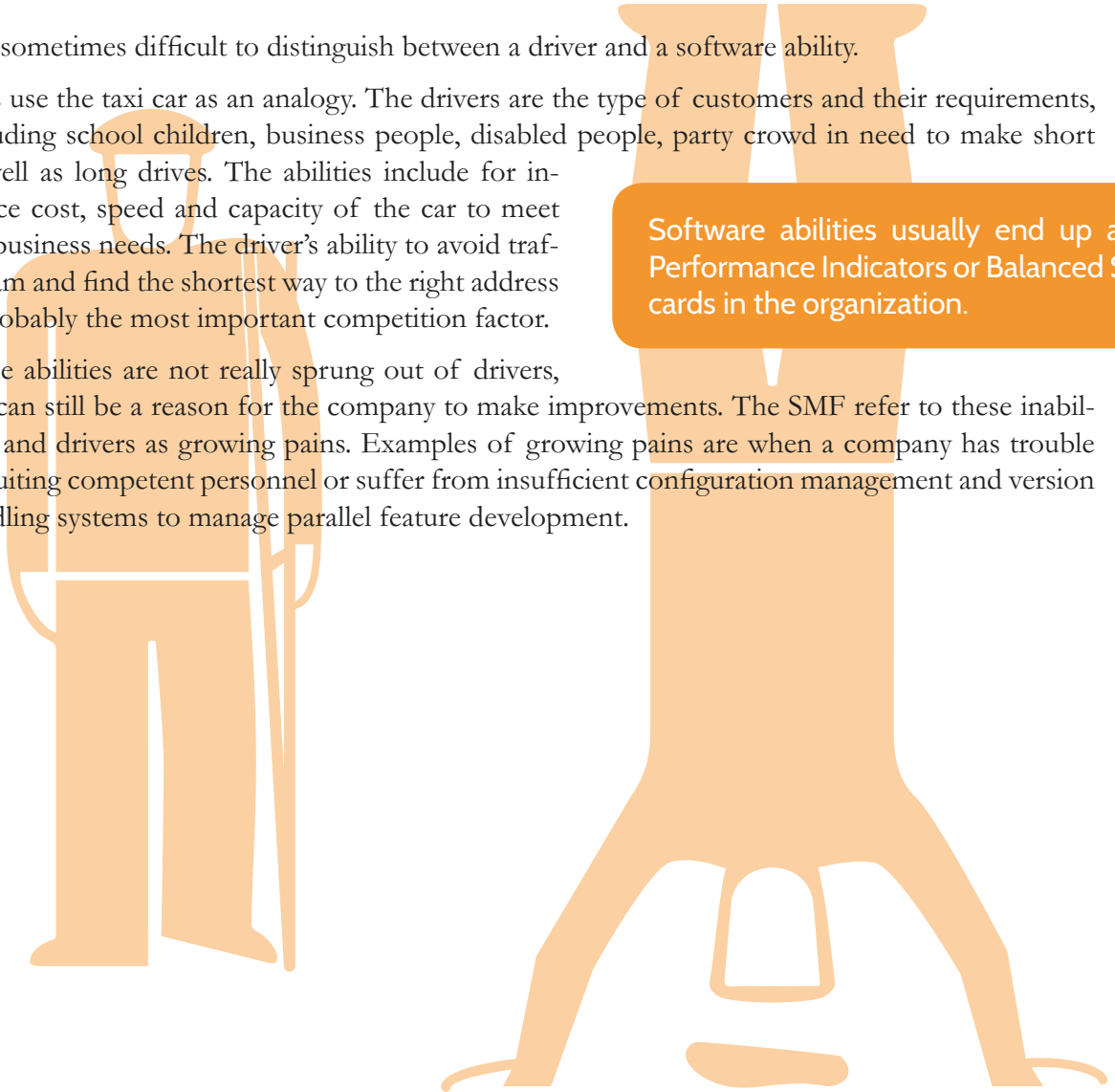
- **Revenue** – How much do we earn from our products or services?
- **Cost** – How much do we invest in development?
- **Speed of development** – How fast is the software being developed? This covers all aspects of speed, from adding or updating a feature to deliver the product or service.
- **Speed 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.

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

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

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

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

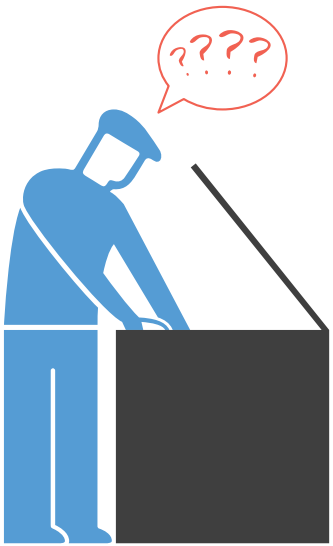


Software model set-up

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

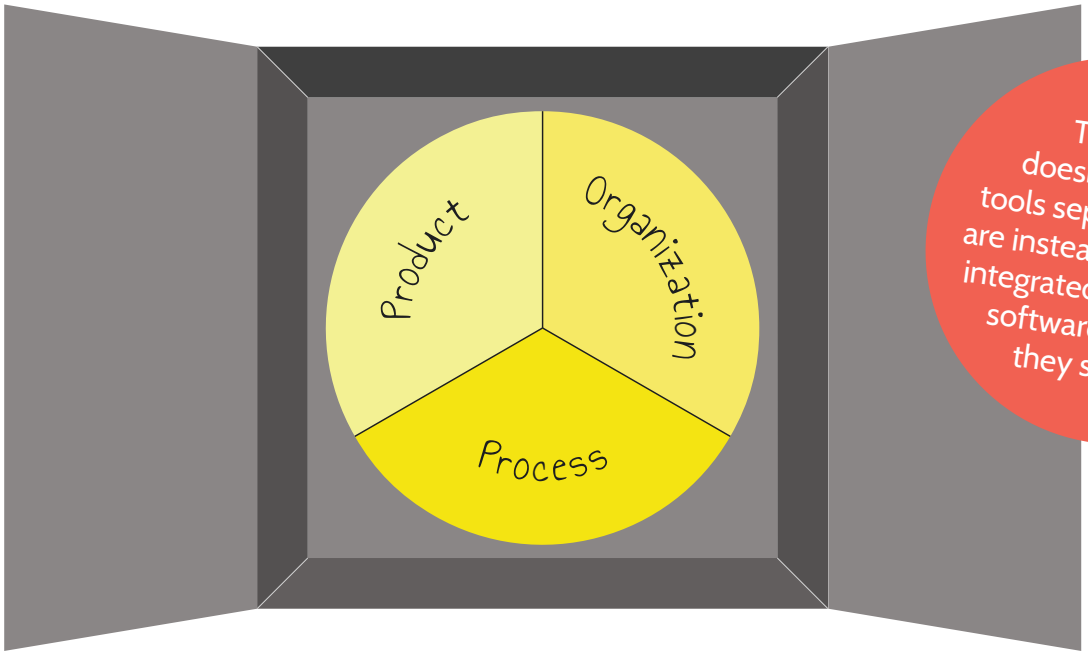
The purpose of the software model domain is to:

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



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

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



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

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

Why? Why? Why?

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

Product domain

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

The SMF divides this domain into three building blocks:

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

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

Process domain

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

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

Organization domain

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

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

Transitions

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

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

Examples of transitions are:

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

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

It all fits together

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

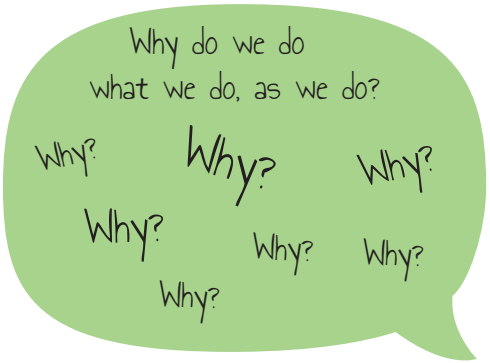
Drivers and abilities gives the structure to describe the reason for scale and the main metrics needed to measure the improvements. Look into case studies with similar drivers and abilities as the business we want to change. Working with drivers and abilities is very important to understand the reason,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

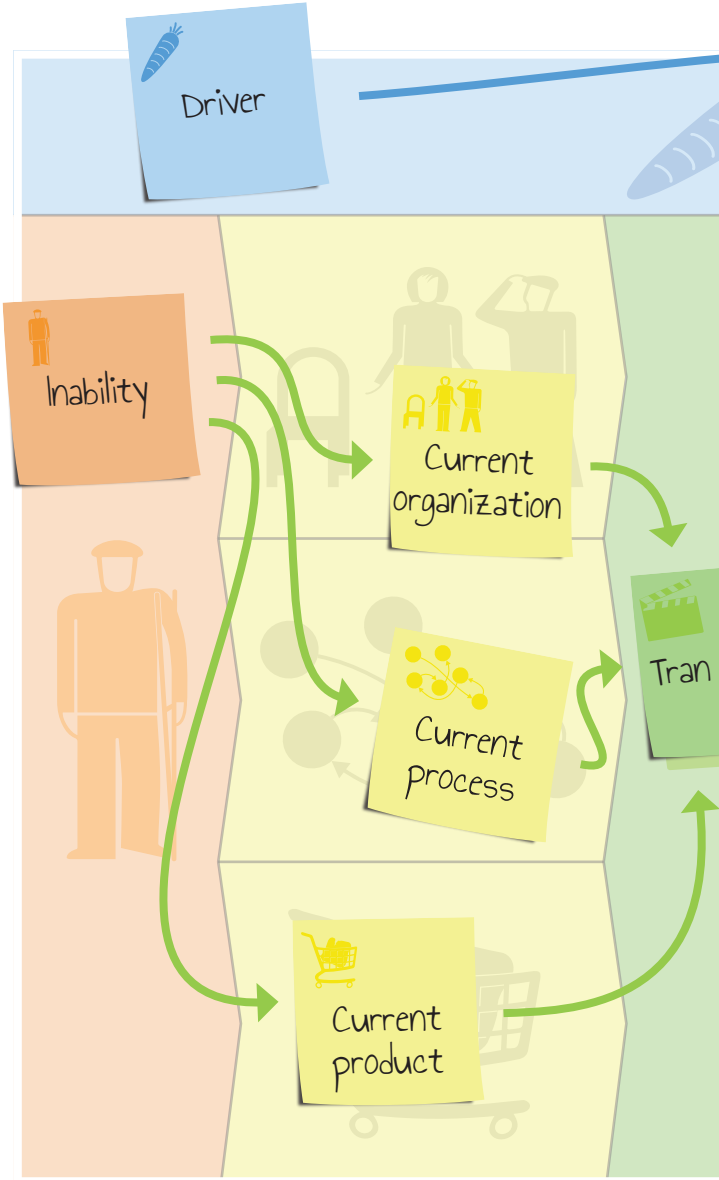
The software model with its three domains and their building blocks gives a structure to describe the actual implementation of the product or service. Each domain studies the building blocks and also the dependencies between implementations in different blocks.

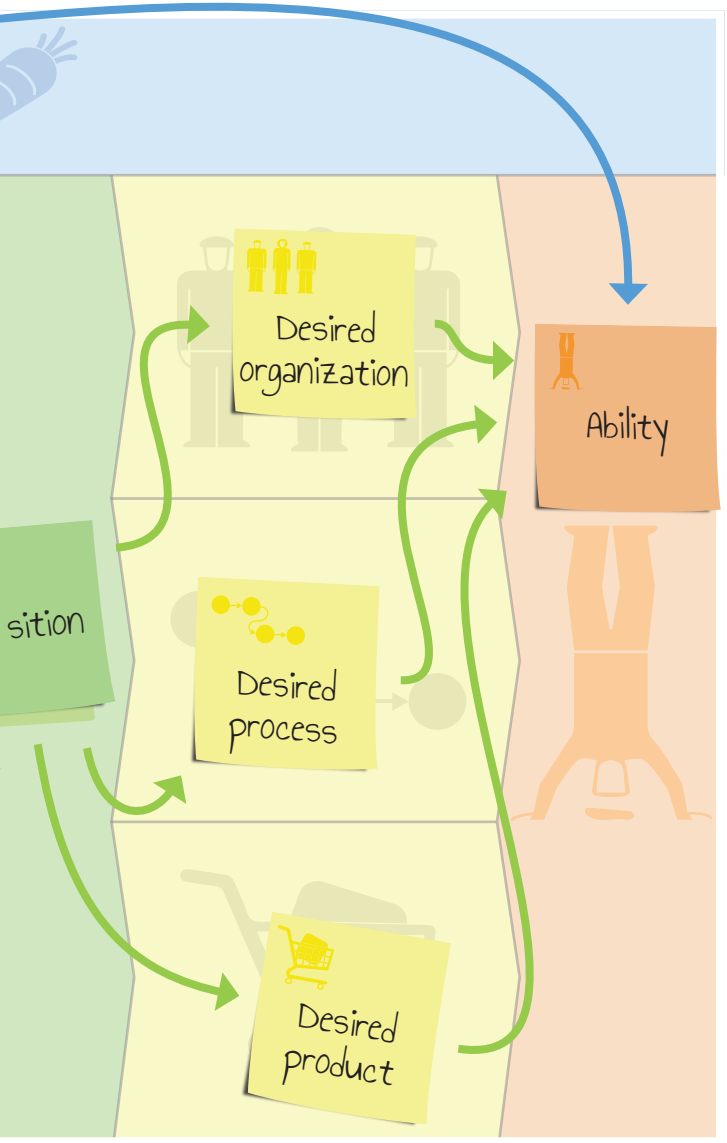
The SMF defines several standard change scenarios, but still most companies have to customize their unique set of changes in order to implement their particular drivers.

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

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


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

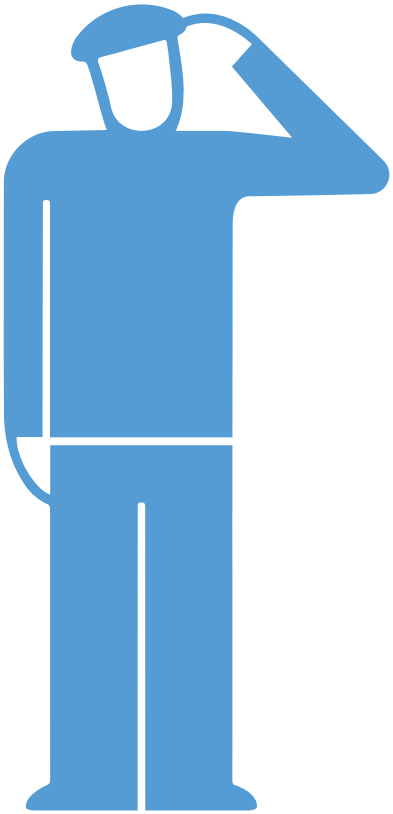

Driver
Inability
Current organization
Current process
Current product
Tran



Why do we want to scale the business?

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

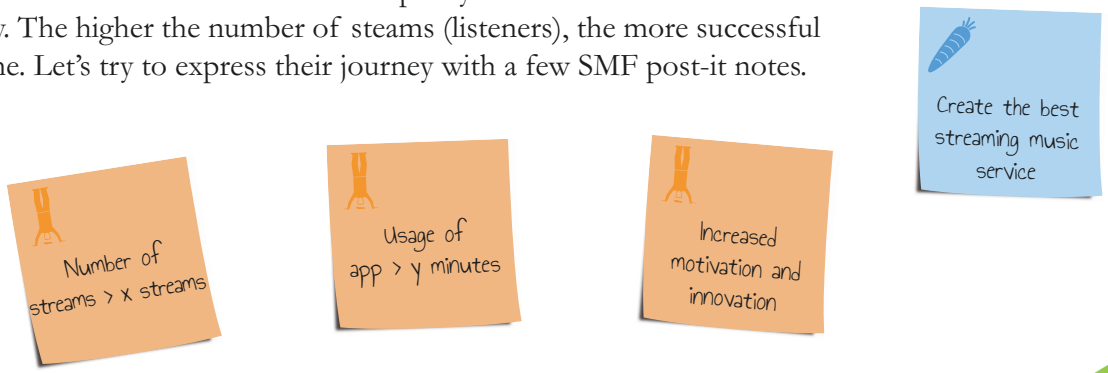
Can we measure the abilities as KPIs?



How the canvas can be used

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

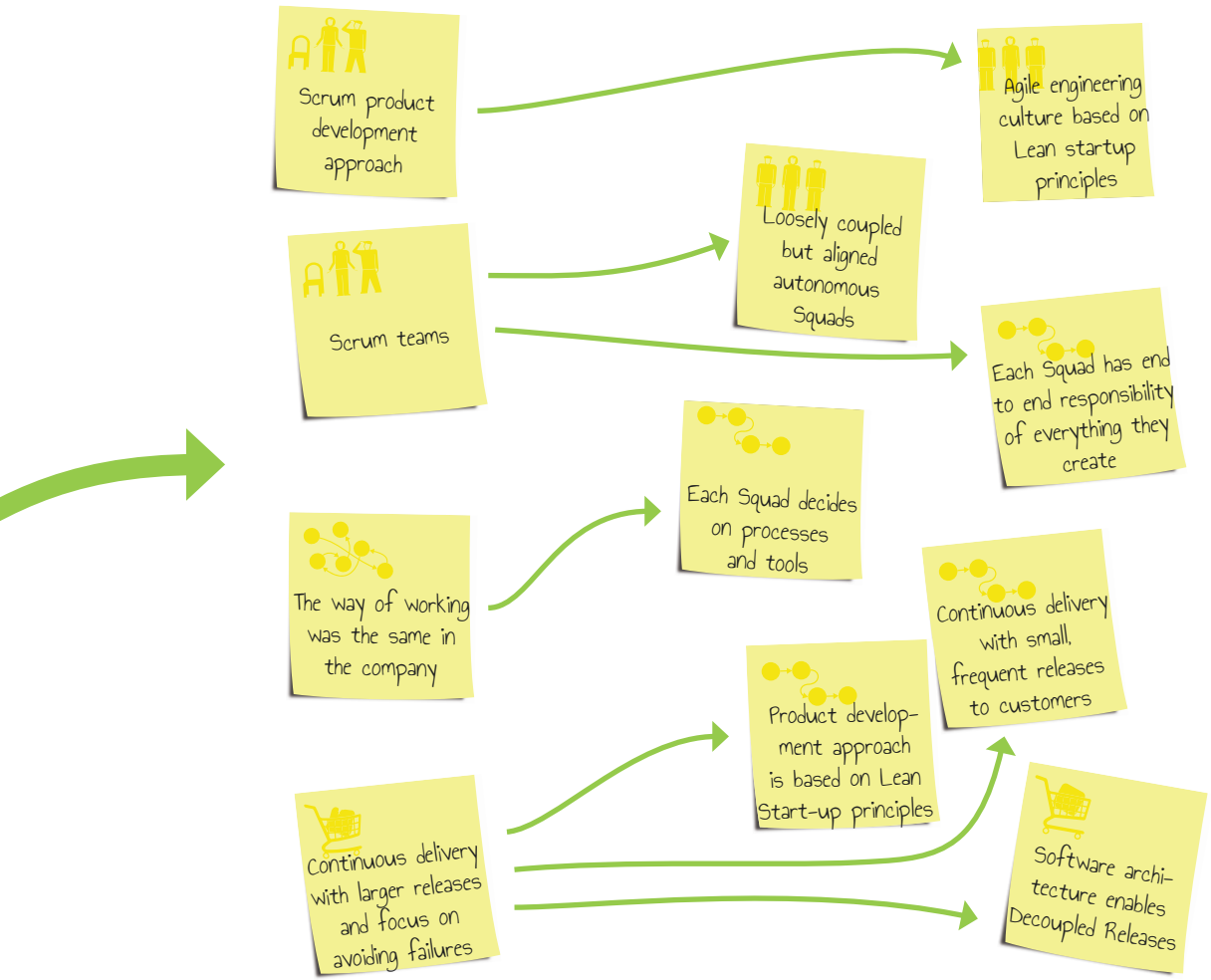
How did Spotify achieve to make the software service so successful? According to the vast amount of articles that have been written on the topic, they simply seem to have decided to "create the best streaming music service". We can assume this was their driver. What they intended to measure was the number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



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

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

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



* The Lean Startup. Crown Publishing Group. Eric Ries

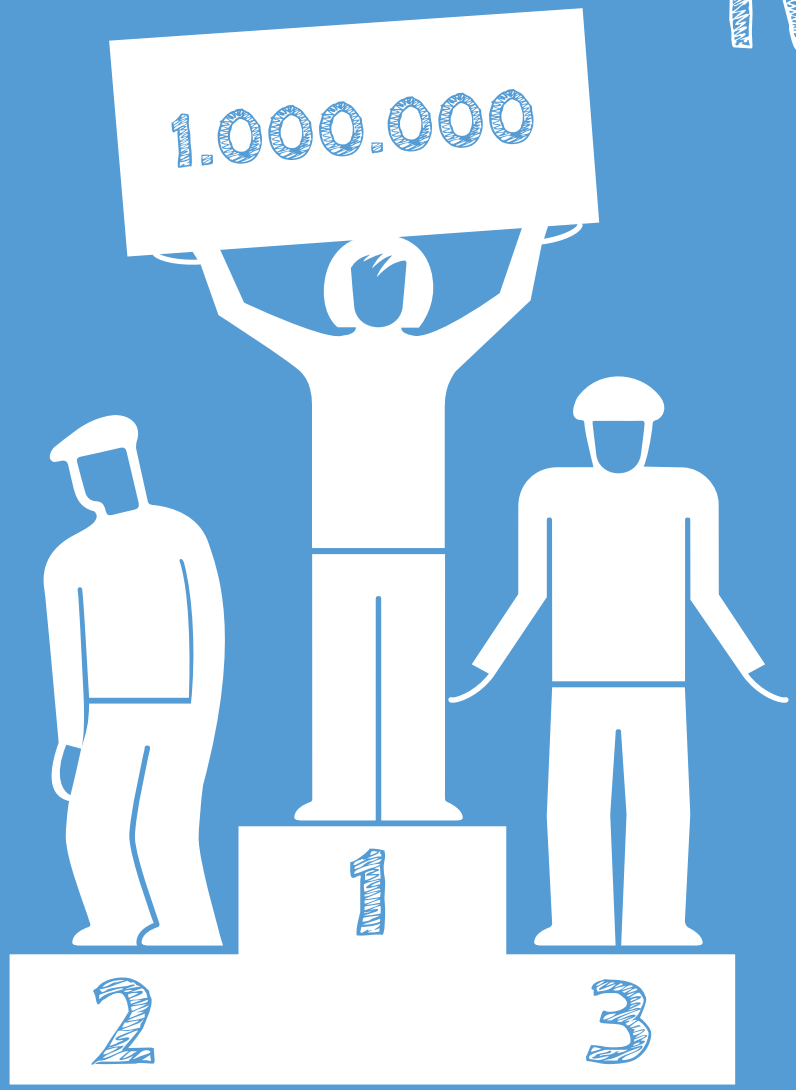
The co

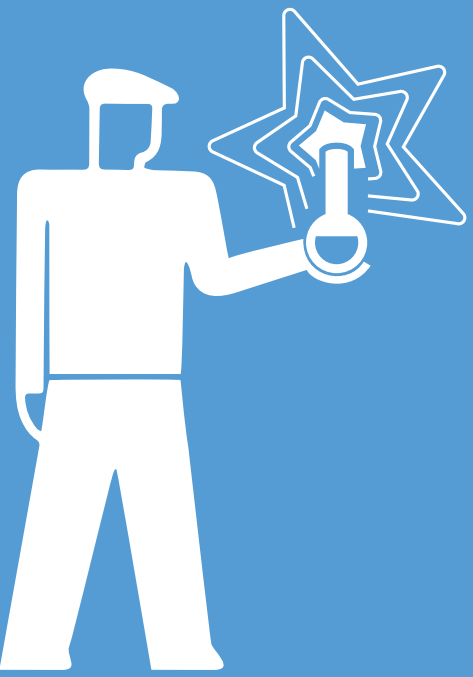
Ways to find

mpass

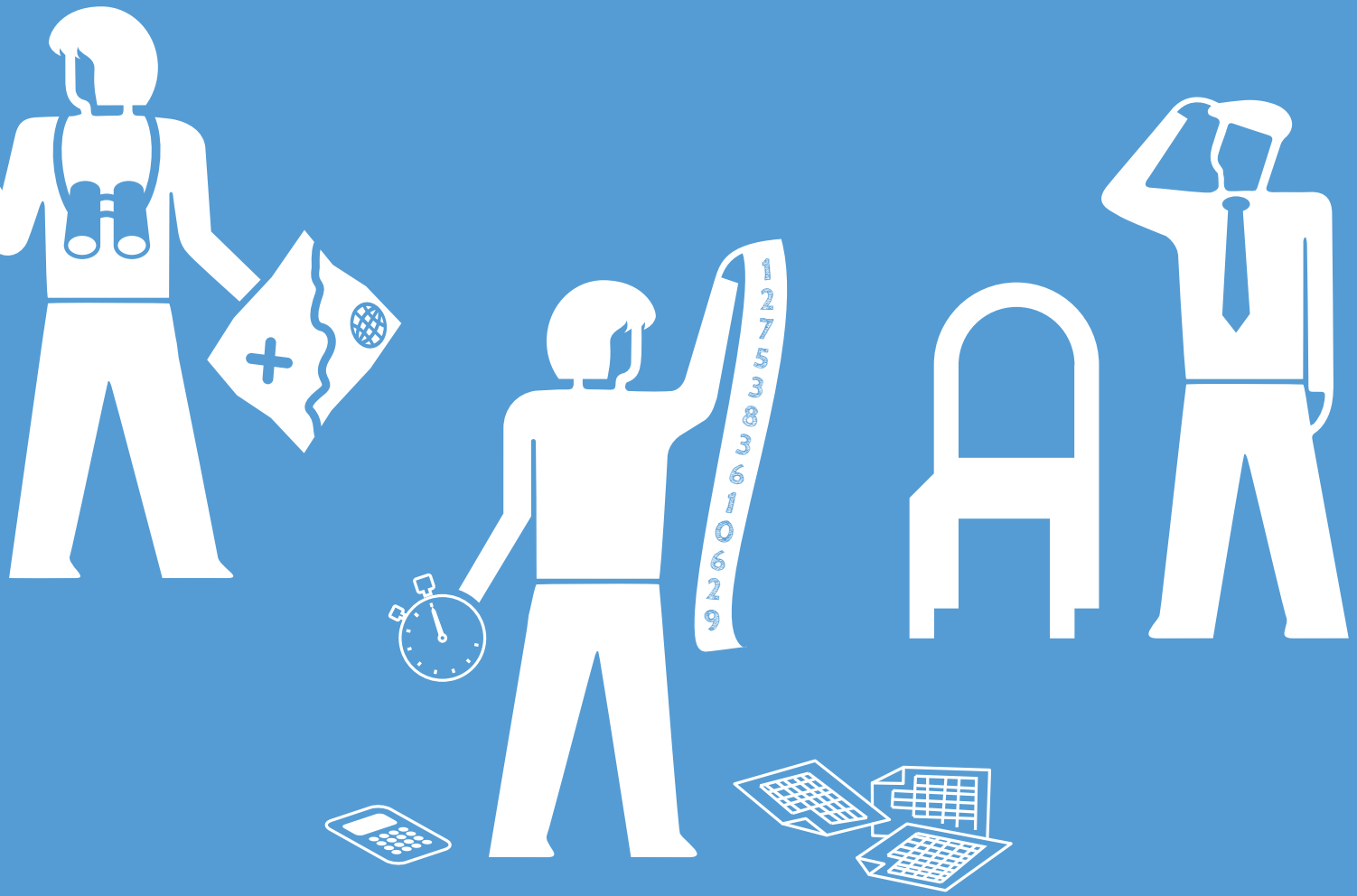
journeys

Two ways to





read this book



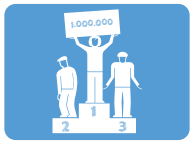
We didn't intend this book to be read from start to finish – but you are more than welcome to do so. Instead, we've written this book with busy managers in mind – people, like you, who don't have the time to read the whole book carefully but only the sections that are relevant to your particular business.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

Journeys

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

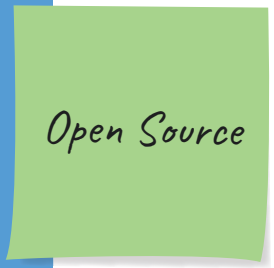
Journey 1
Journey 4
Journey 5
Journey 6
Journey 8

Deal with leadership challenges

Journey 5
Journey 8

Journey 1 - Co-operate in a community

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

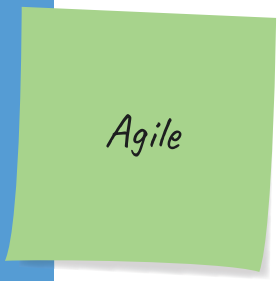


Journey 2 - Building ecosystems

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

Journey 3 - Add supplementary services

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



Journey 4 - Deliver 24/7

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

Agile

Journey 5 - Pump up the volume

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

Agile

Journey 6 - Agile and disciplined

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

Offshoring
Outsourcing

Journey 7 - Outside the box

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

Basic
software
engineering

Journey 8 - First things first

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