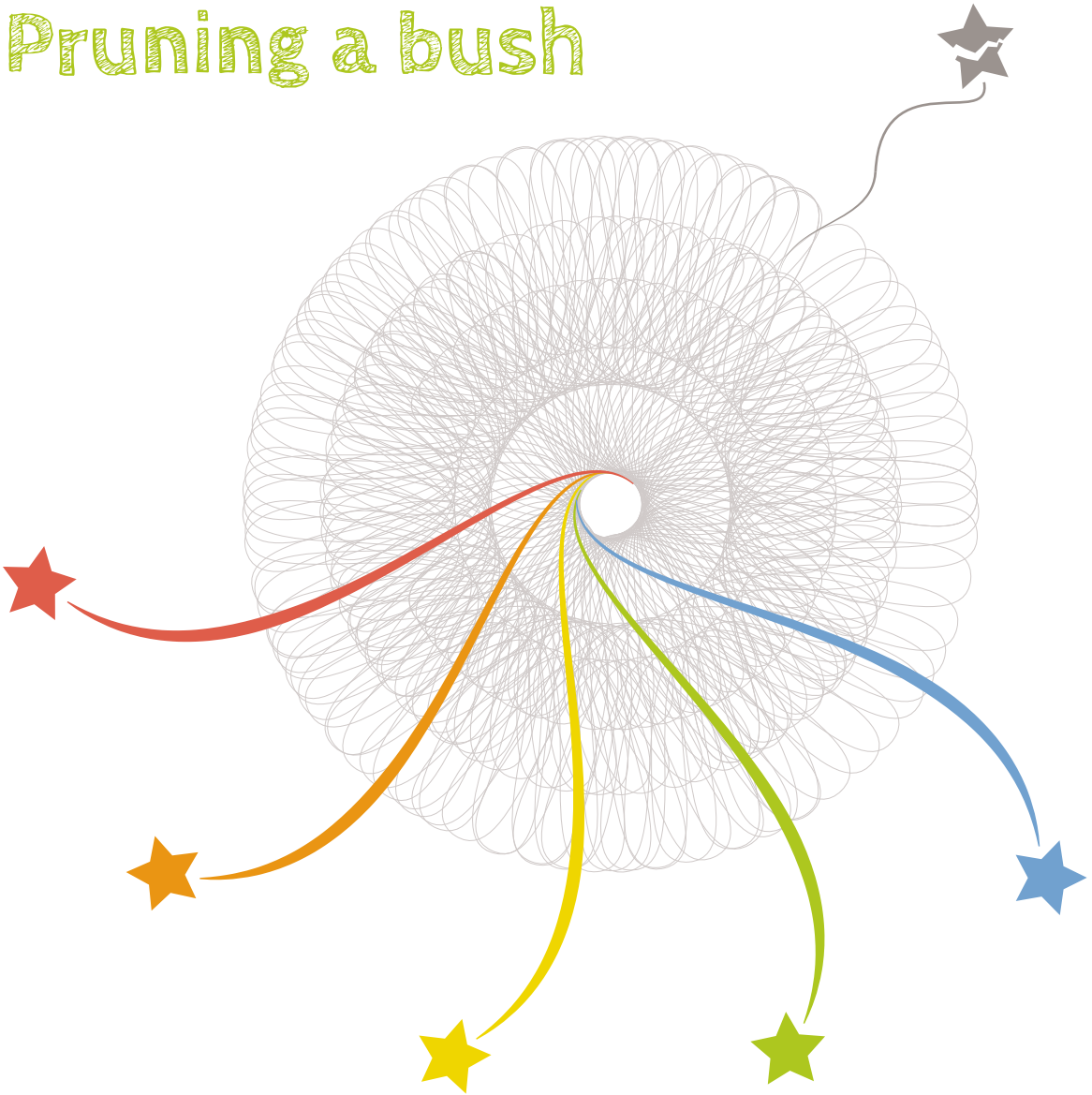
CASE STUDY / Deliver 24/7

Pruning a bush



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

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

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

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

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

* www.scrumalliance.org
kanbanblog.com/explained

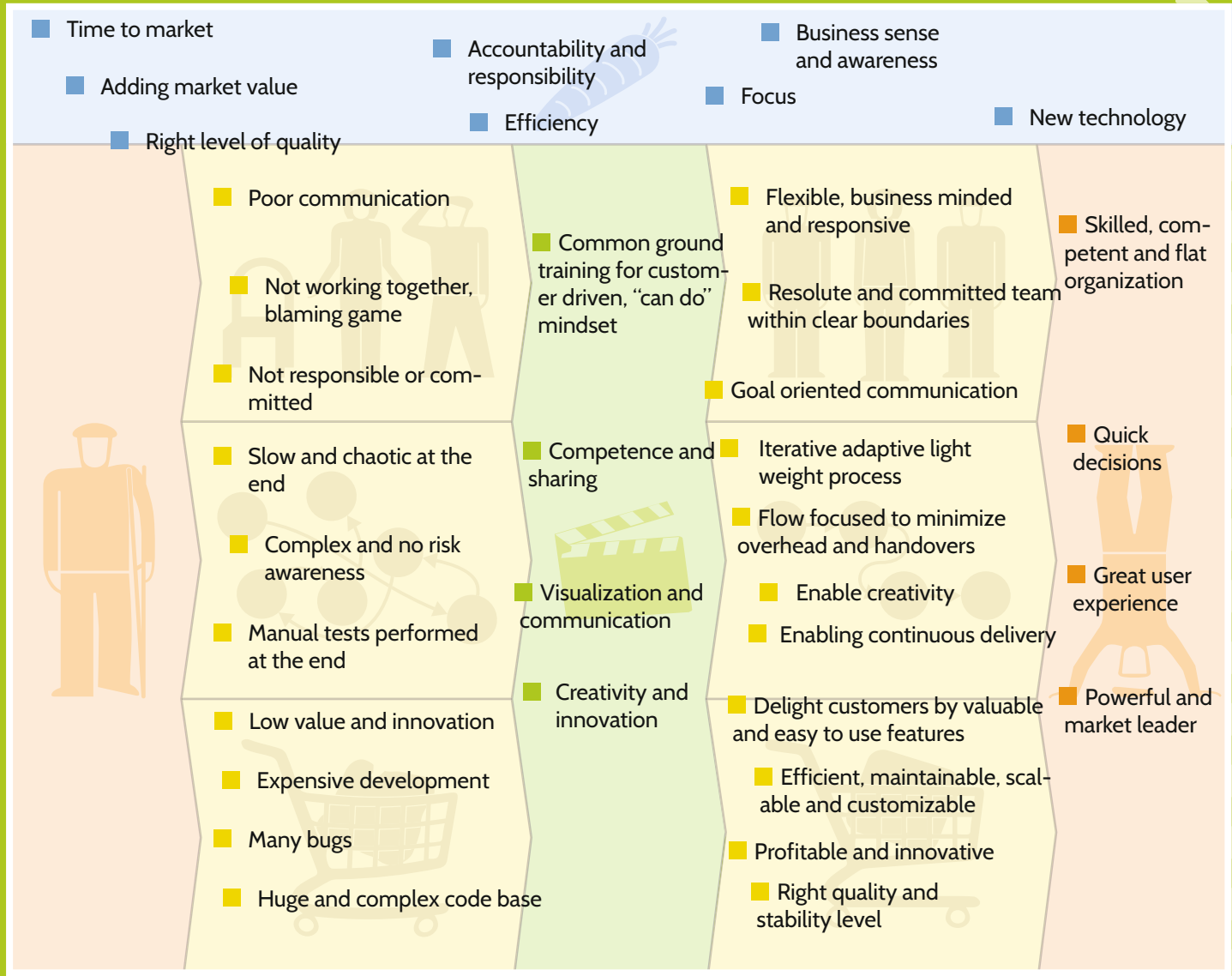
ucts started to work according to Scrum. The people in the test team were all distributed in the agile teams. They were assigned to write user stories, define acceptance criteria and also to train the developers in how to test. The testers could now focus on more complex test cases and the overall quality.

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

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

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

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



CASE STUDY / Deliver 24/7

Ensuring prima deliveries

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

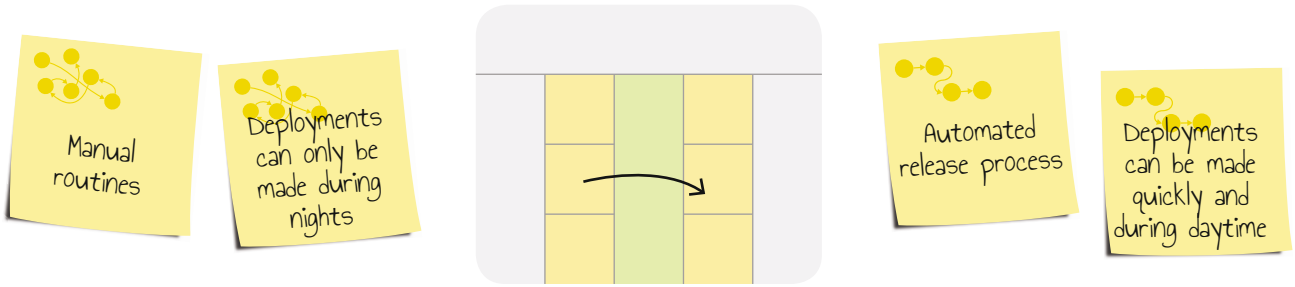
This was also the situation when the Swedish company *Projekstyrning Prima* (Prima) decided to implement Continuous Delivery of its route planning systems. To that end, Prima developers introduced new systems for source control management, build automation, automated deliveries, and a new way to log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

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

Cloud platform Microsoft Azure

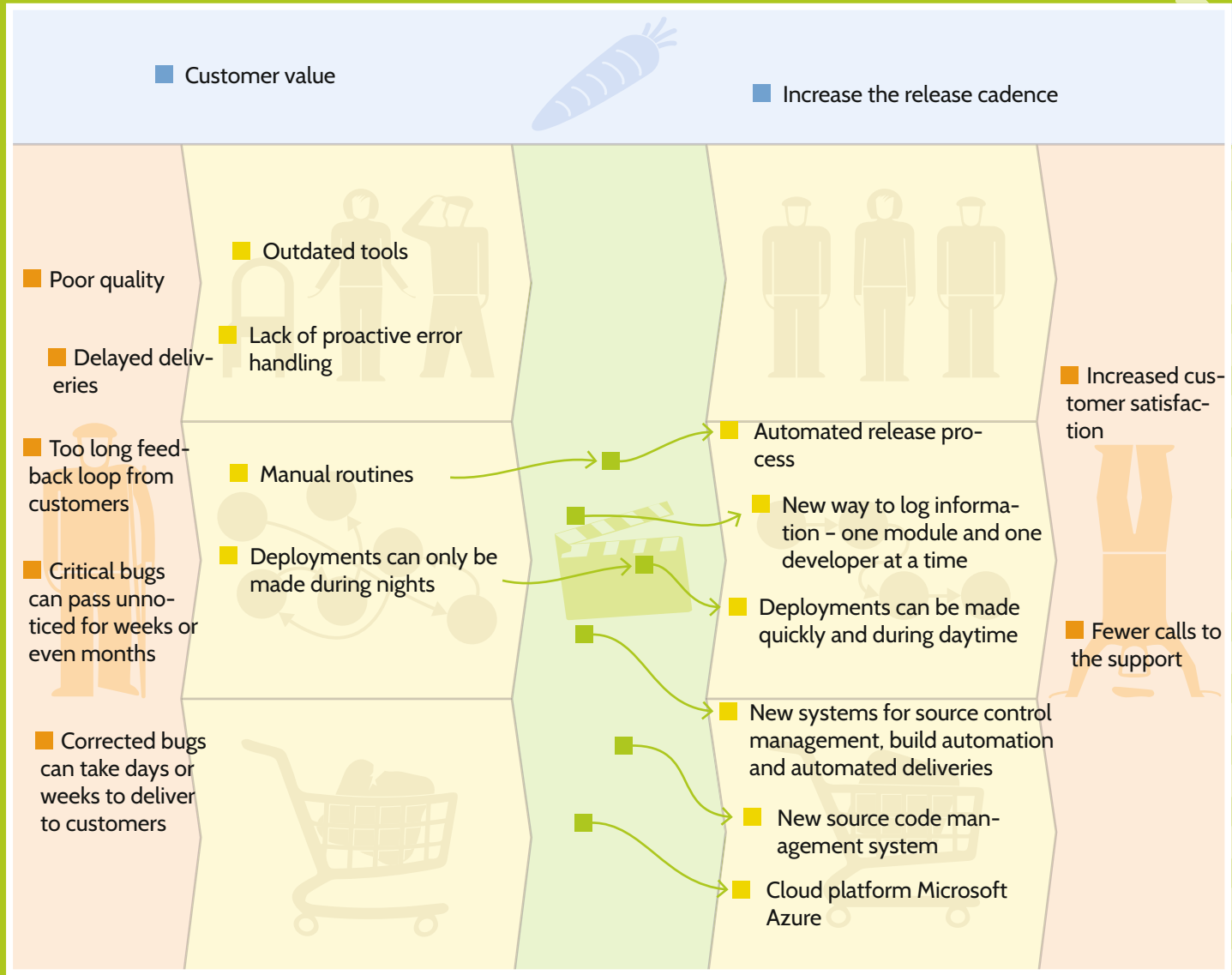
Prima was using the Microsoft Azure cloud platform, but this wasn't a key problem. The challenges they ran into could have arisen with any cloud technology stack. While cloud technology certainly facilitates a transformation to continuous delivery, this wasn't a key requirement.

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

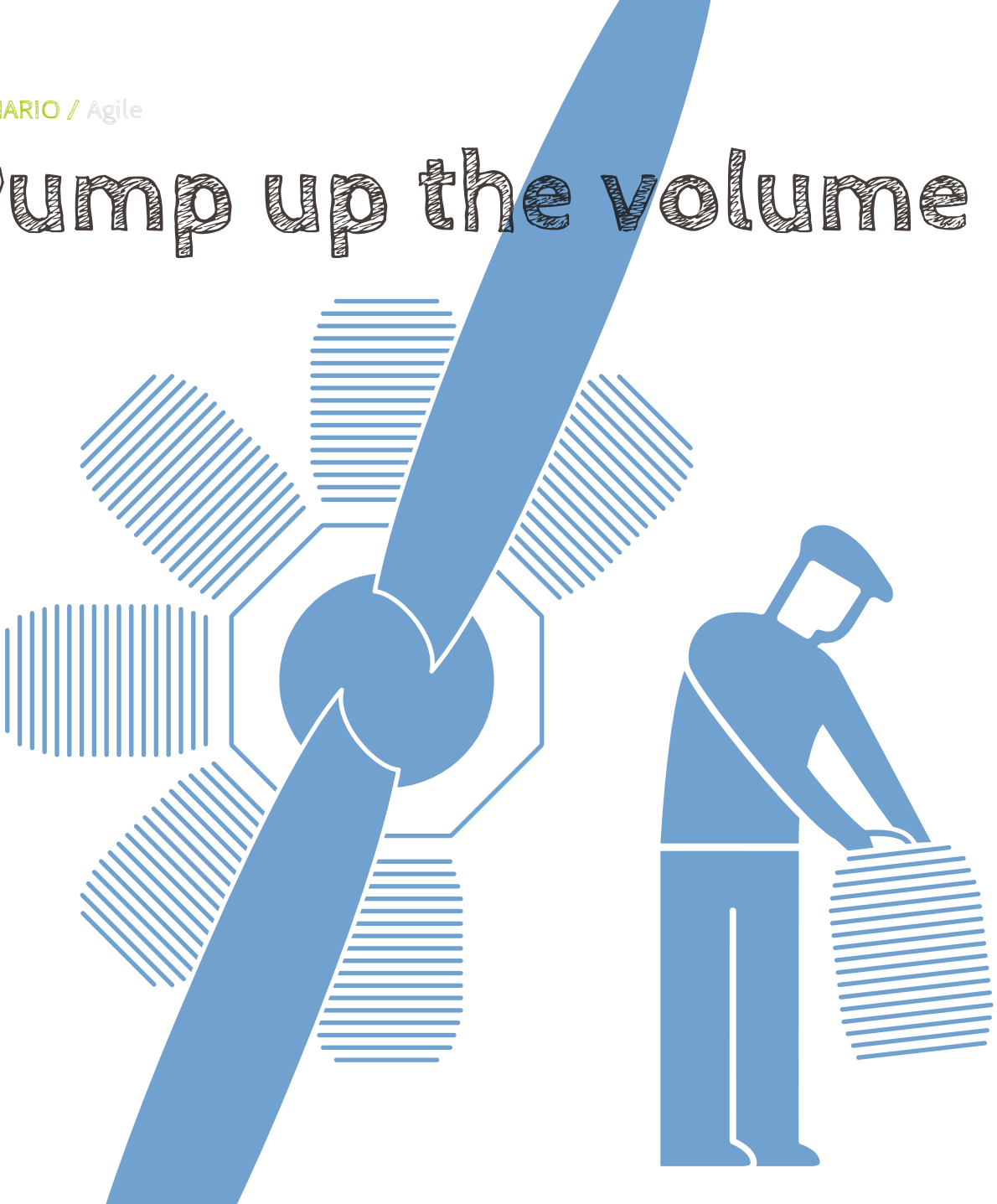


New source code management system

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



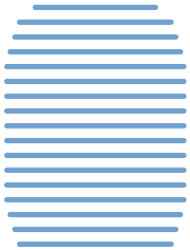
Pump up the volume

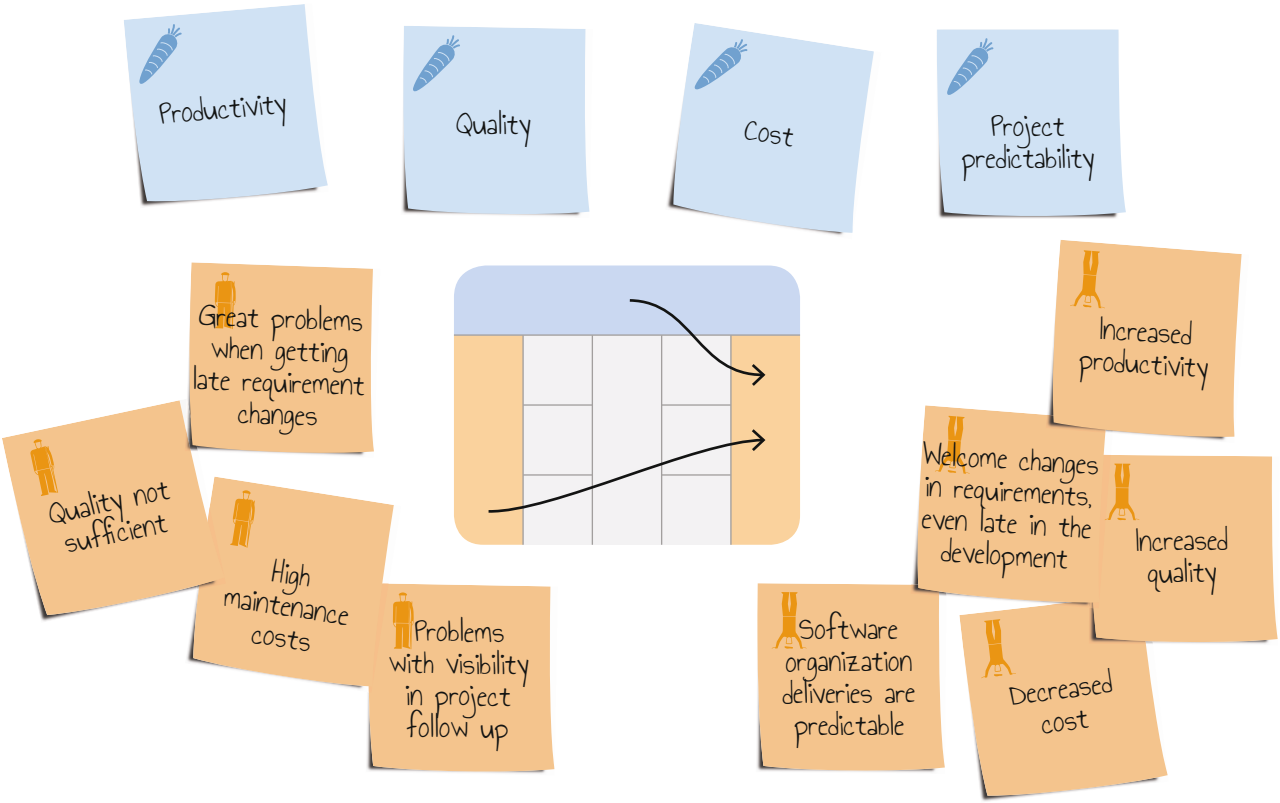


> The scaling Agile model is for companies that want to start working with Agile on a larger scale. The complete model describes scaling in three domains: size, offerings and value stream.

This chapter focuses solely on the size domain.

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





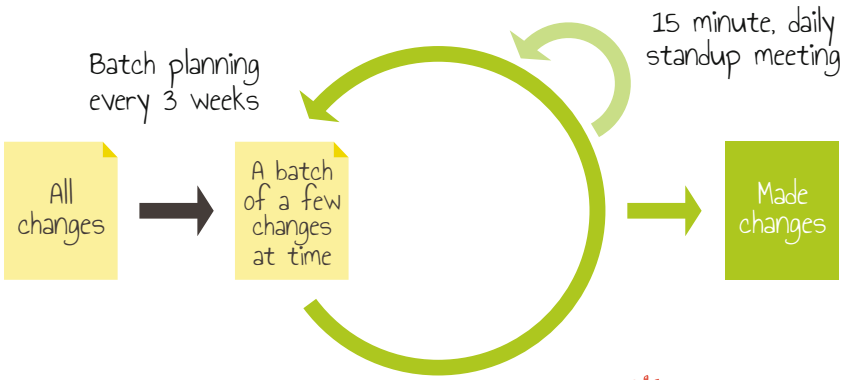
When scaling a business in terms of efficiency, all that matters is how to build the products faster, more predictable and in bigger volumes. Innovation is not a matter at this point. This is a business need that mostly appears in very large companies that deliver complex products or services to the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

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

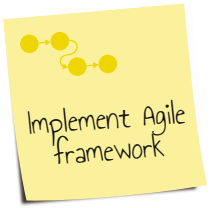
Agile development, for example Scrum and Kanban*, has rapidly proven to be the preferred methodologies among software teams and their developers. Even if a company formally hasn't made the switch to use Agile methods, it's not unlikely that some of their teams already have started to work this way. Teams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

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

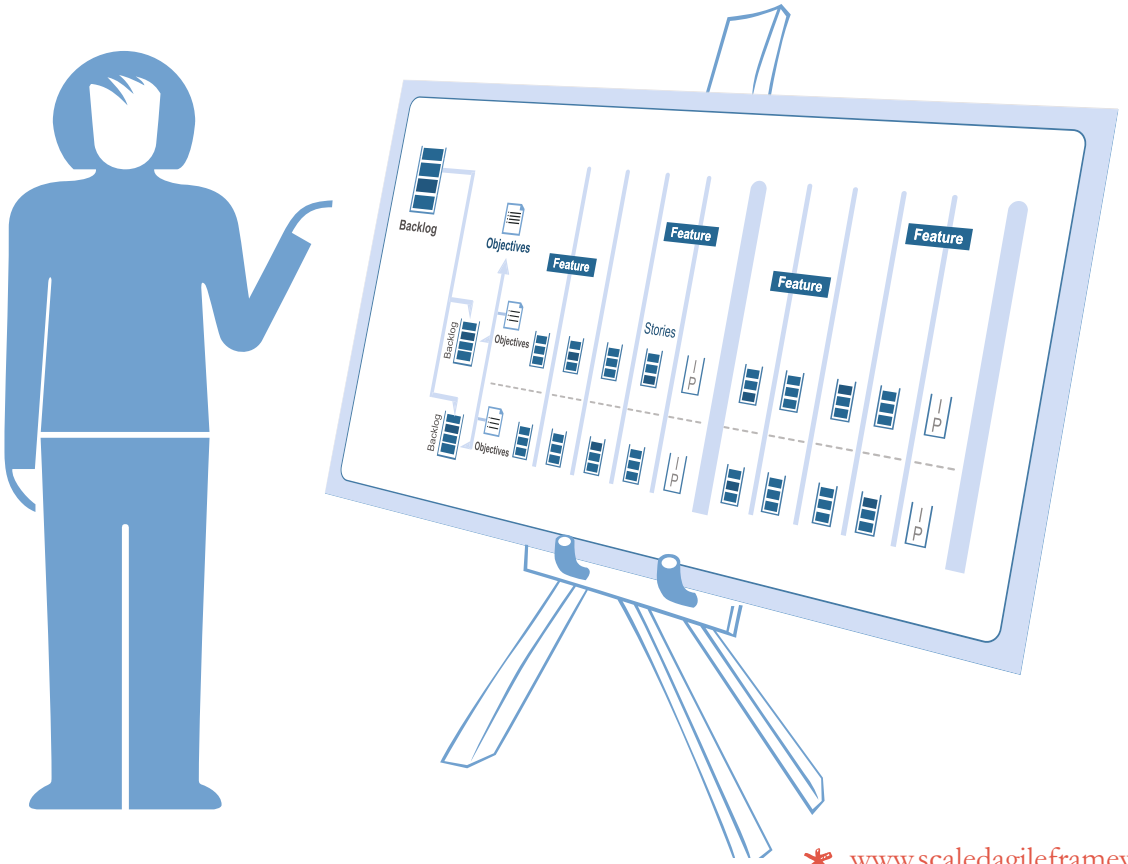
Organizations that have been scaled in this way shows that they are able to deliver utterly complex products and services with remarkable efficiency. There are of course pitfalls ahead, when focusing on efficiency over a too long period of time. While being reactive is considered good in Agile, it's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



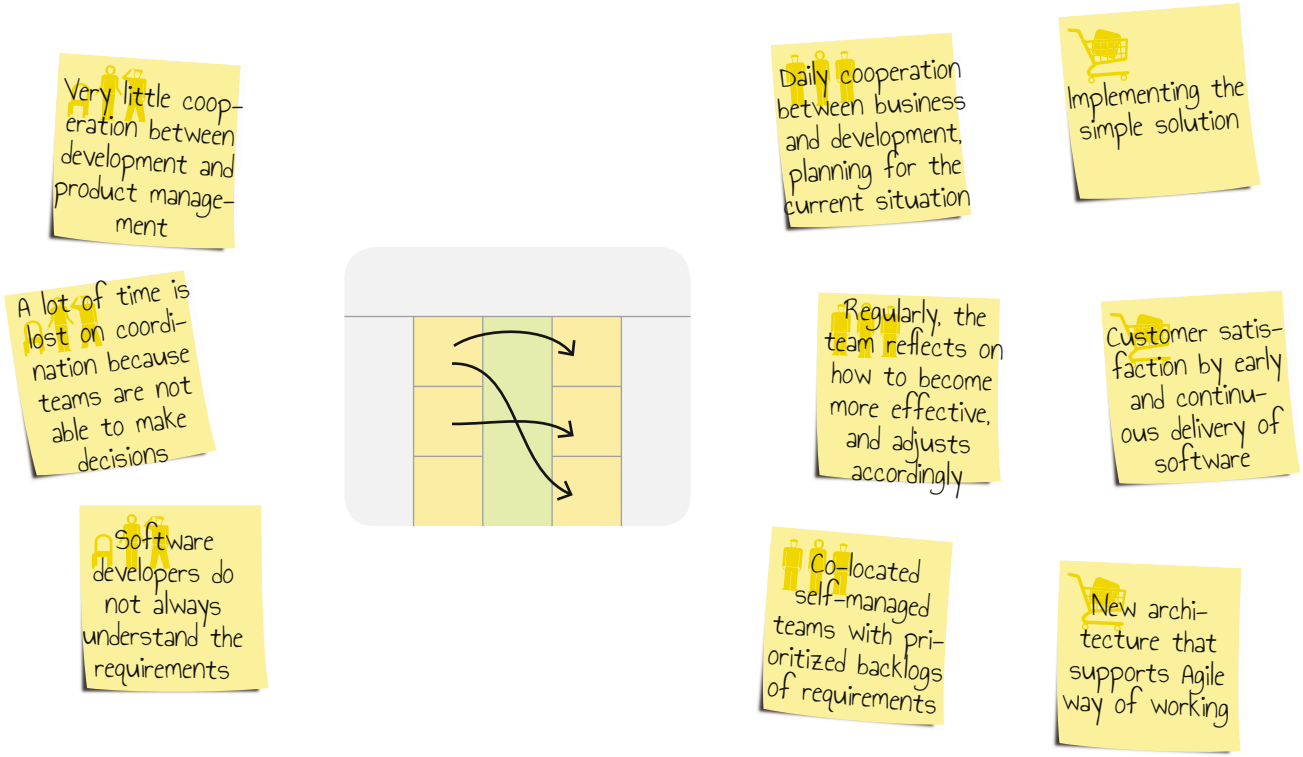
* www.scrumalliance.org
kanbanblog.com/explained



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

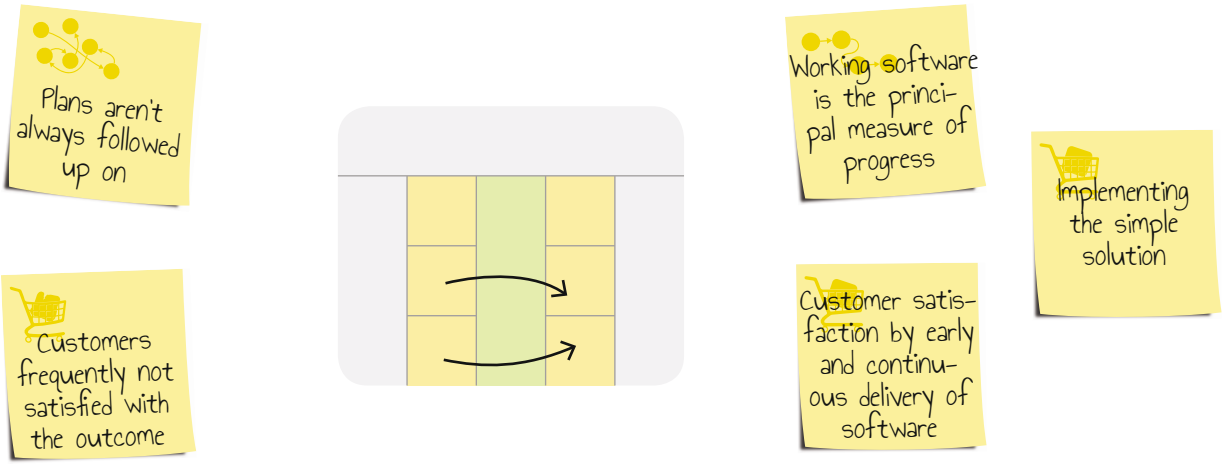


* www.scaledagileframework.com



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

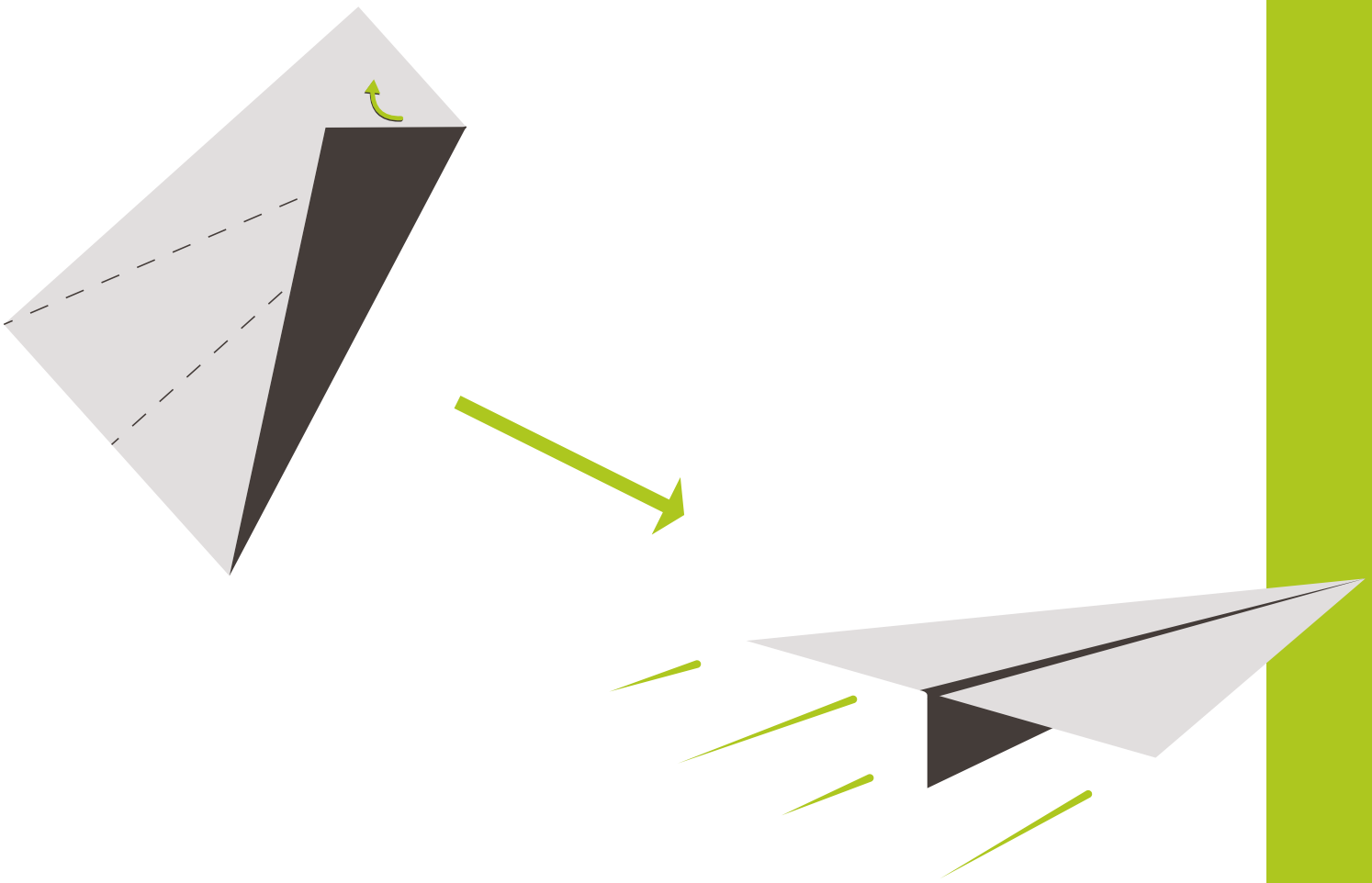
self-managed co-located cross-functional teams working. The teams should understand the requirements fully and have all necessary competence to take all decisions for the functionality that they are responsible for. Read more about change in **Your journey**.

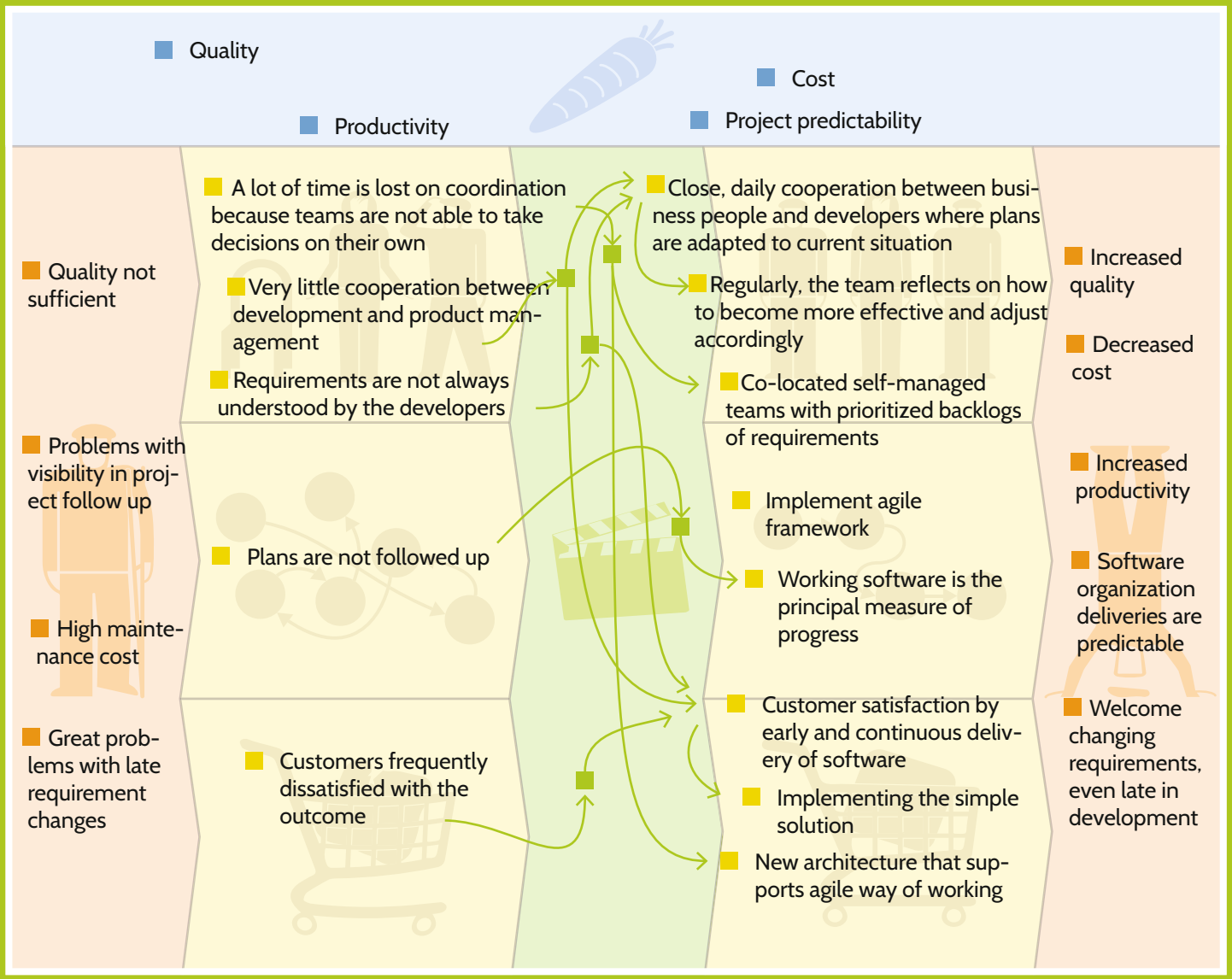


Another important aspect is to have close cooperation between the product management and the development team. Ideally product management competence must be in the self-managed teams, so that the communication is fast and direct. But there are some Agile frameworks that promote to have the product management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

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

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




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

Get inspired

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

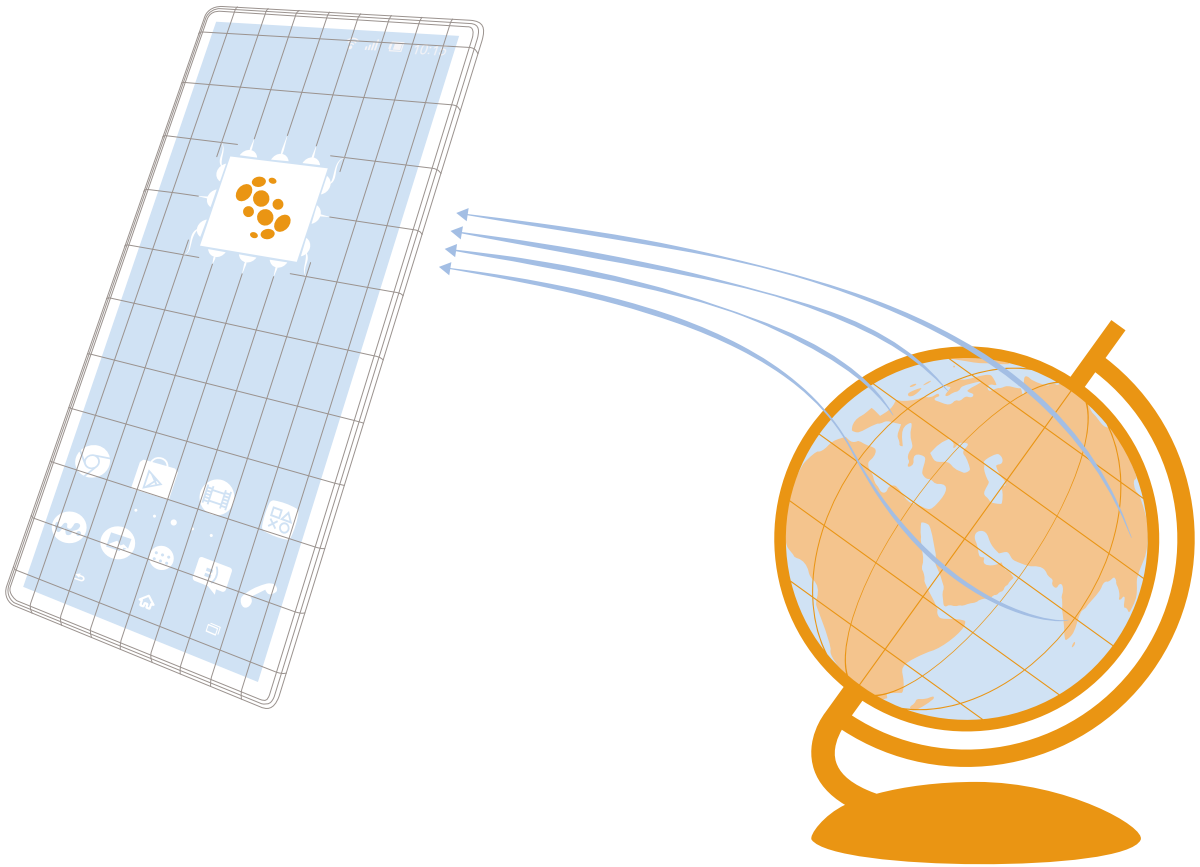
Global R&D goes agile with SAFe

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

Multi-site development

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

Global R&D goes agile with SAFe*



* www.scaledagileframework.com

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

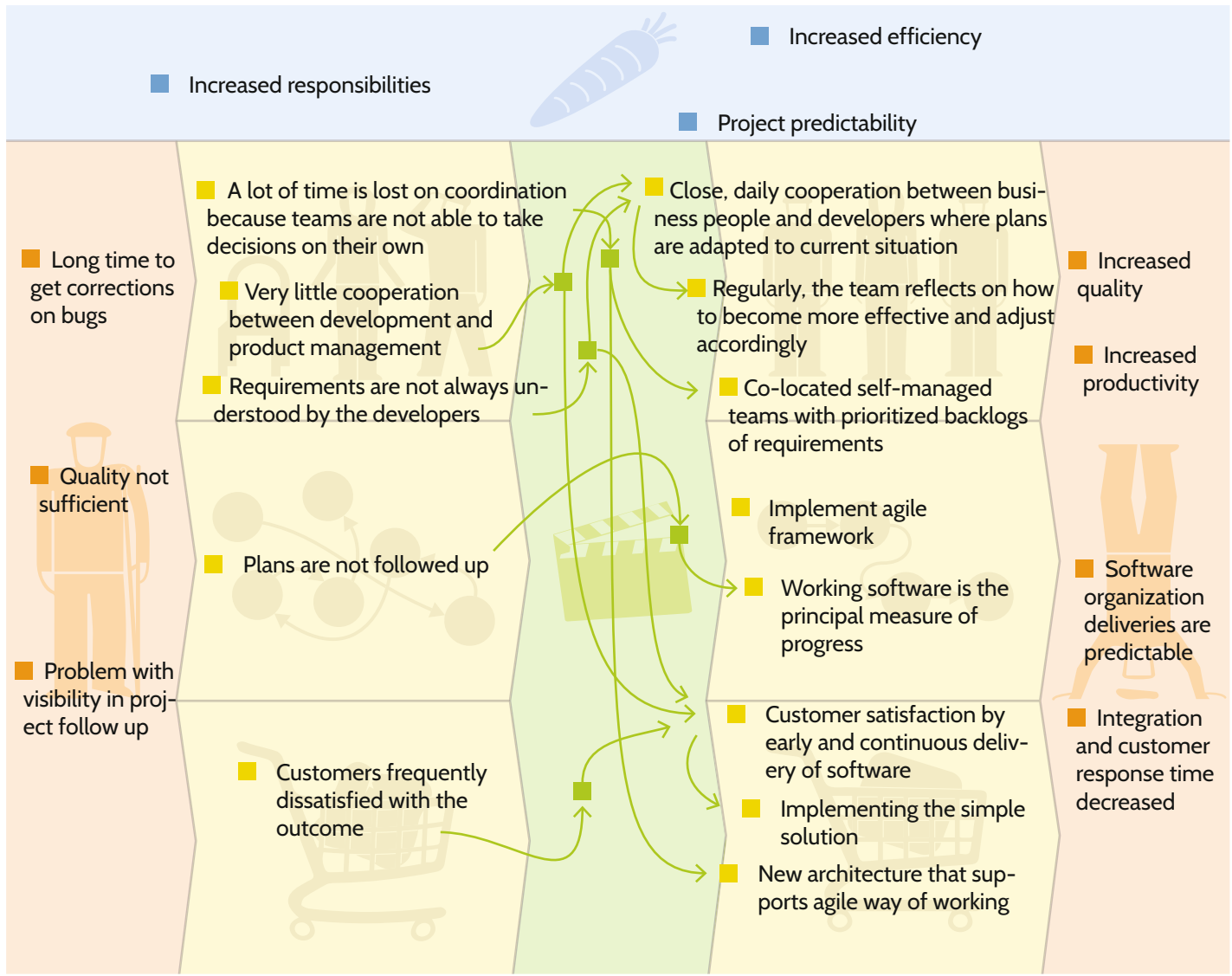
These drivers initiated their Agile transformation:

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

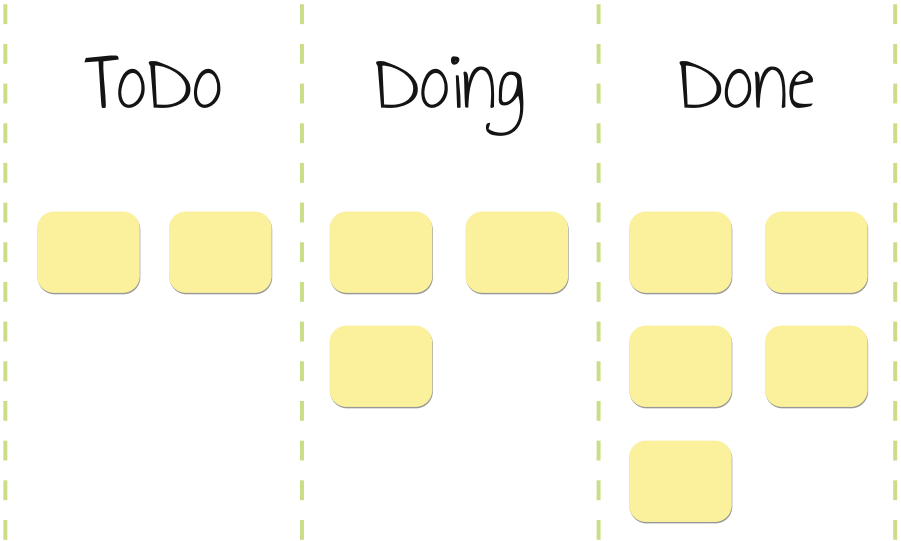
The organization was set up as an R&D organization parted in four requirement areas (RA) and one integration & verification organization. Each RA was responsible for either a technology area (like core SW) or a function area (like test). These RAs were located to 4 different sites over the world; in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

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

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



Multi-site development



Productivity

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

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

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

Quality

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

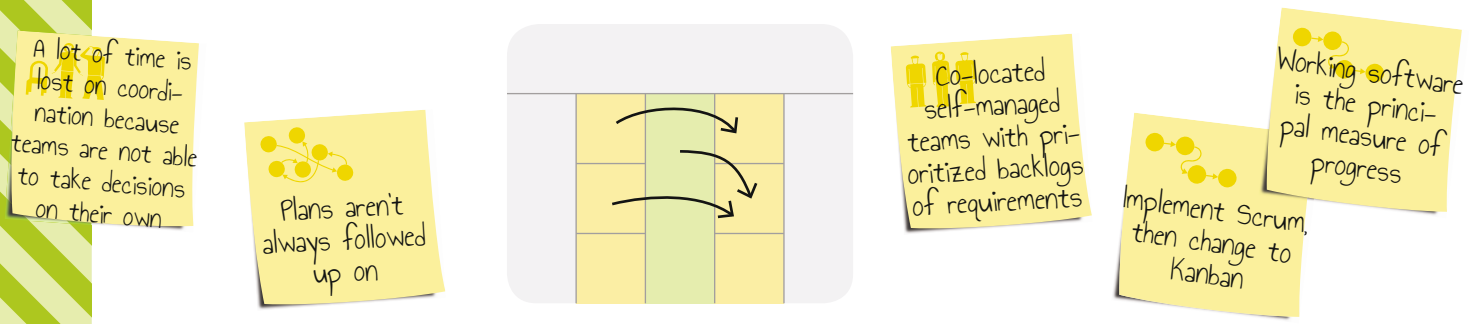
Project predictability

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

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

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

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



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

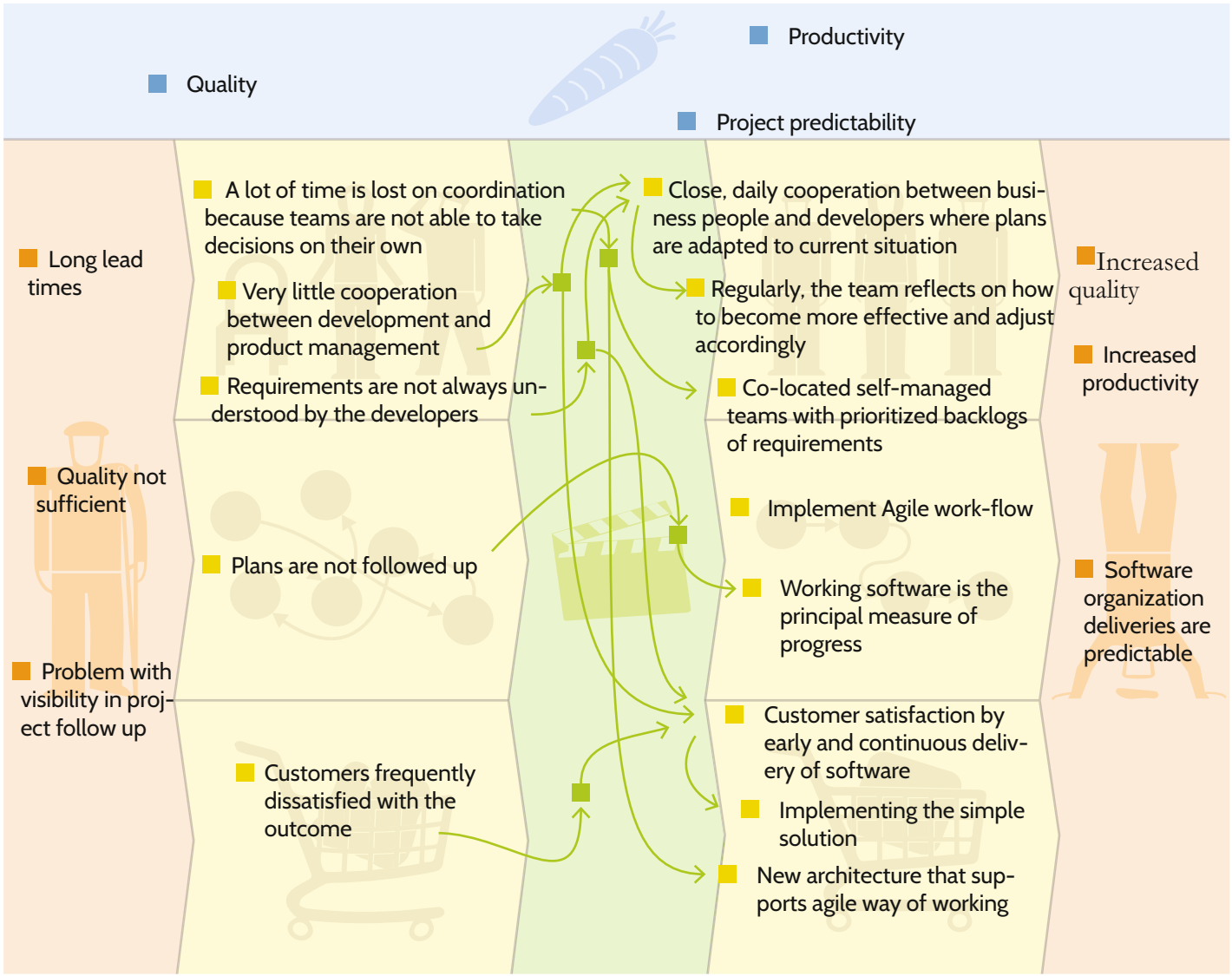
A key get-away from the project was the importance of visualizing work in progress on both project and team level, to make sure problems are detected fast. They solved it by start using JIRA* in the cloud, by this enabling smooth access by both external and in-house teams.

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

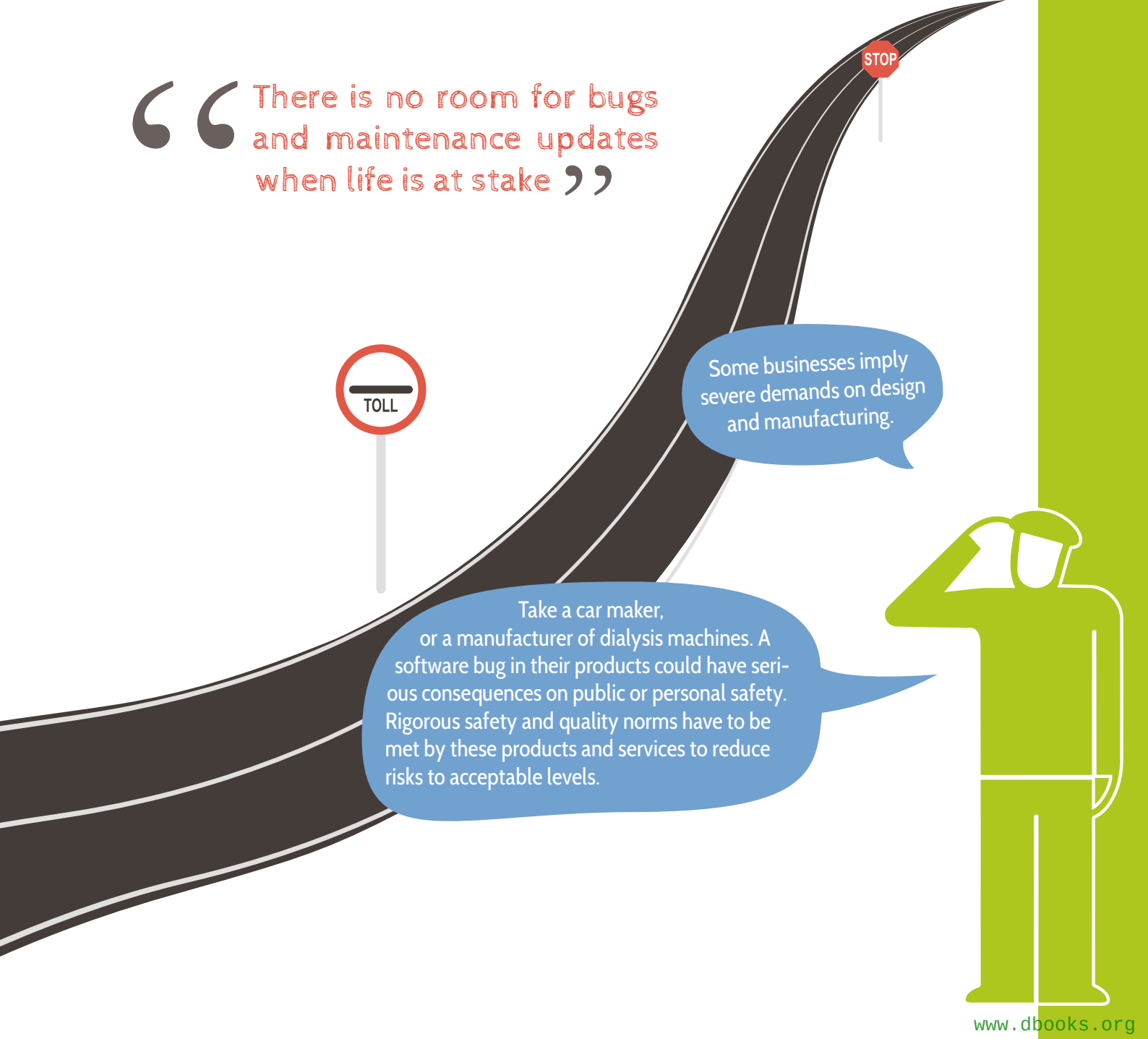
A key motivation for the change was to make their very large project finish before deadline. They desired to be able to predict what part of the scope could be completed by that time. By being able to do this, they could then re-prioritize early and solve problems as they appeared.

Did they deliver in time? Yes, they did.

* A bug tracking, issue tracking and project management tool



Agile and disciplined



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

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

Drivers

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

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

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

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

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

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

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

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

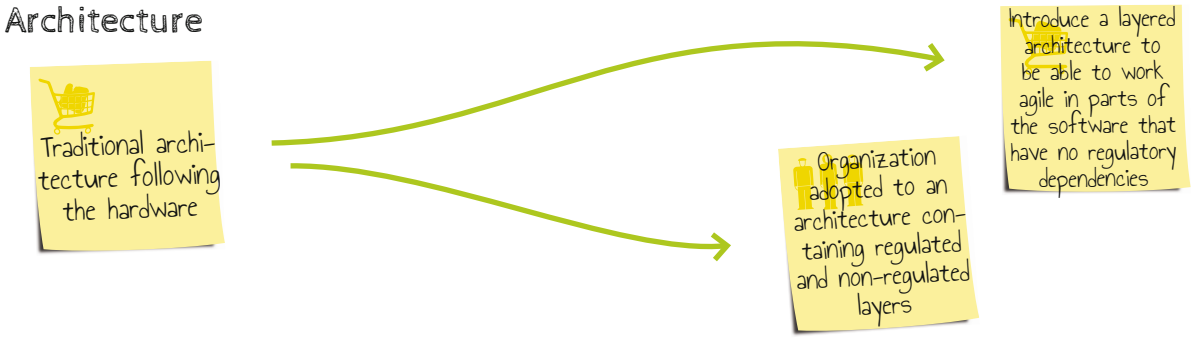
- They shall be satisfied
- They shall be maintained throughout requirement decomposition
- They shall address the software contribution to system hazards
- Hazardous behavior of the software shall be identified and mitigated
- The confidence shall be managed in relation to the system risk

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

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

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

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



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

Autonomous teams

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

Working code, short development cycles and continuous integration

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

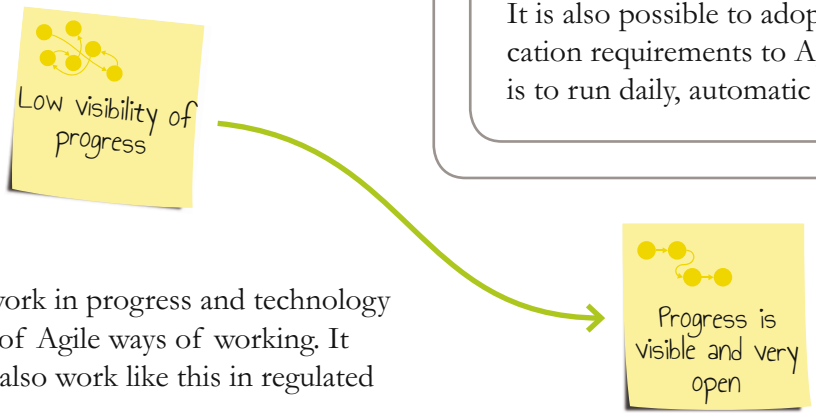
Minimum of documentation and functionality

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

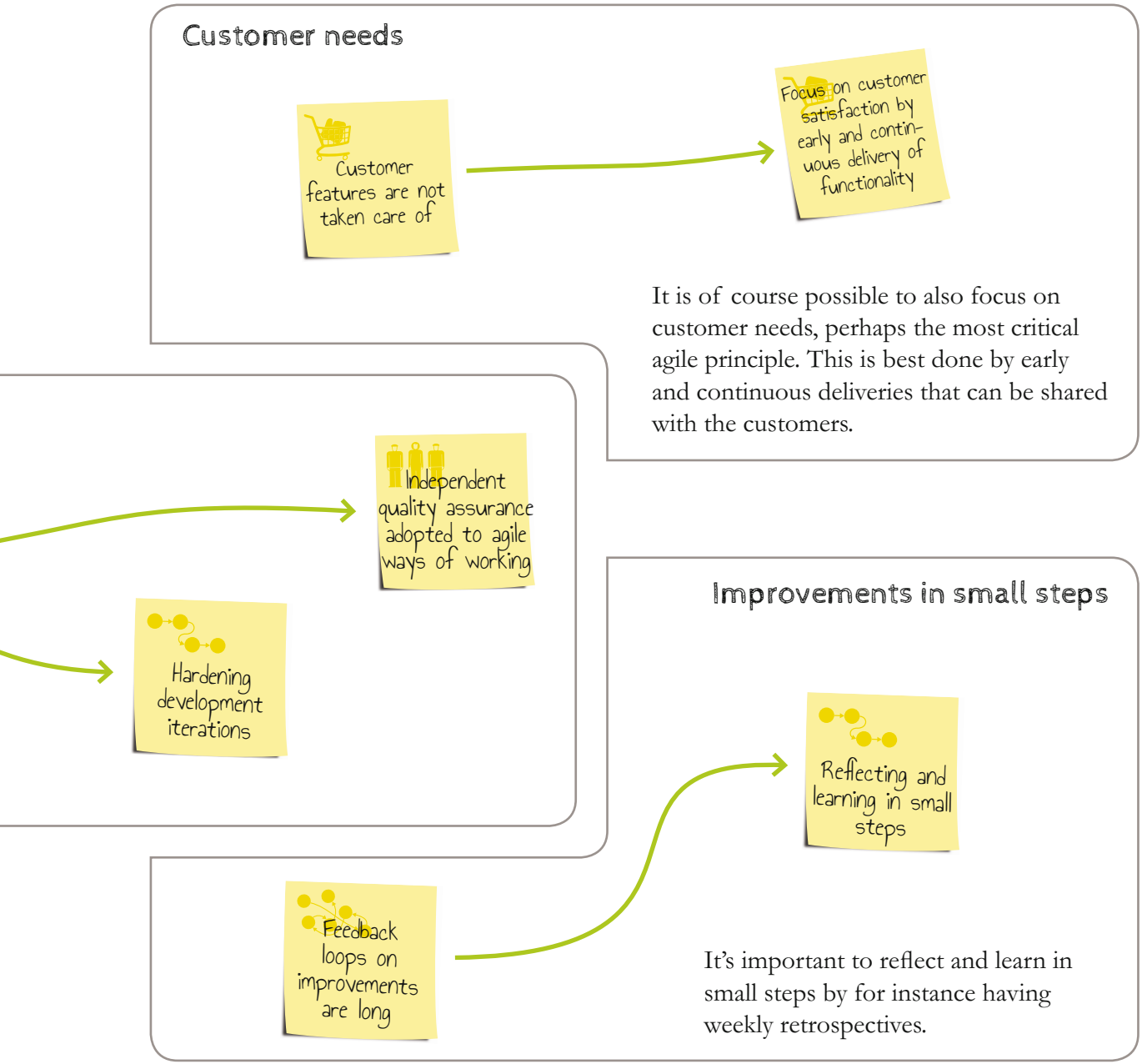
Quality and verification by agile means

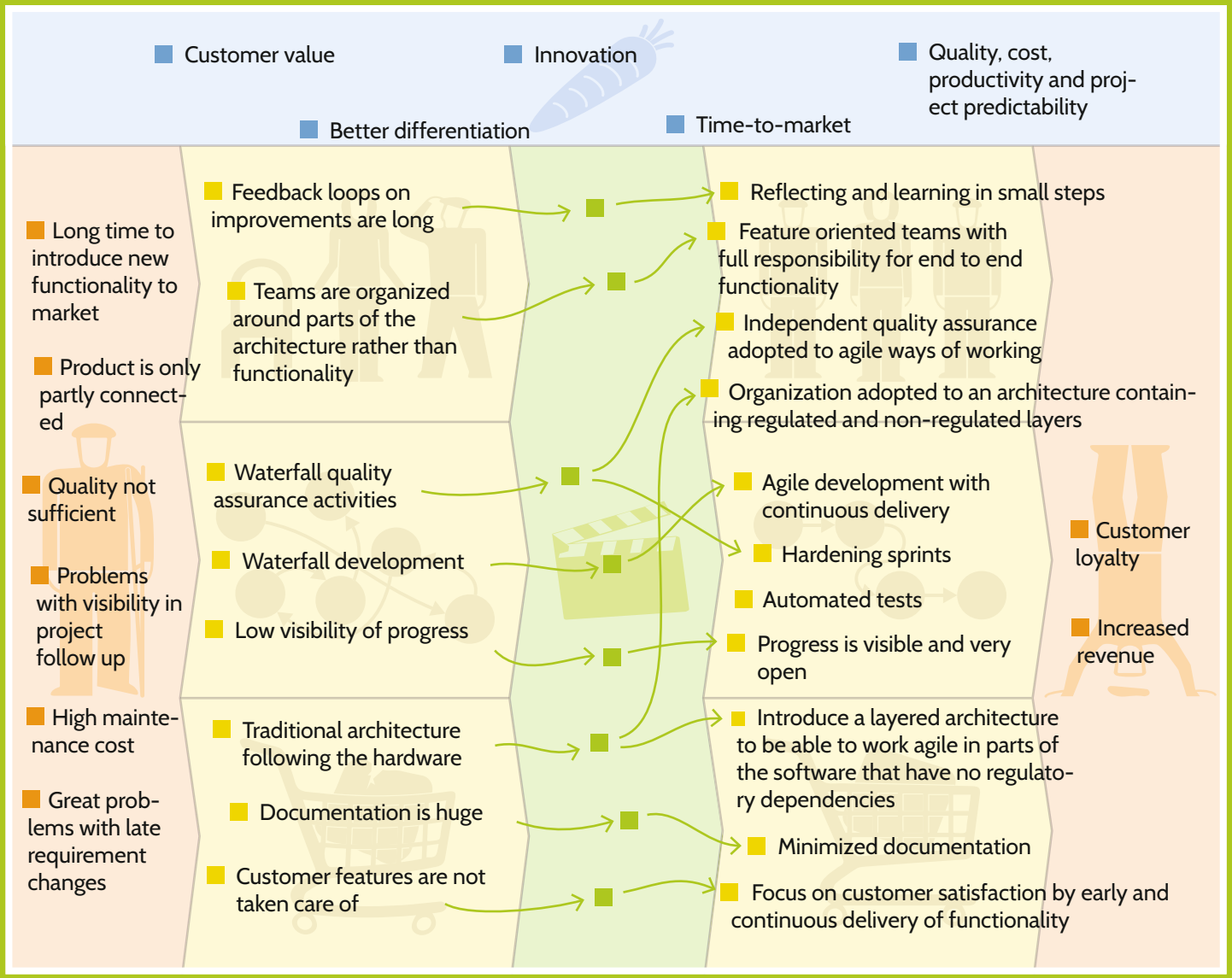
It is also possible to adopt quality and verification requirements to Agile ideas. One way is to run daily, automatic regression tests.

Visualized system and work in progress



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




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

Get inspired

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

Scaling Agile in Automotive

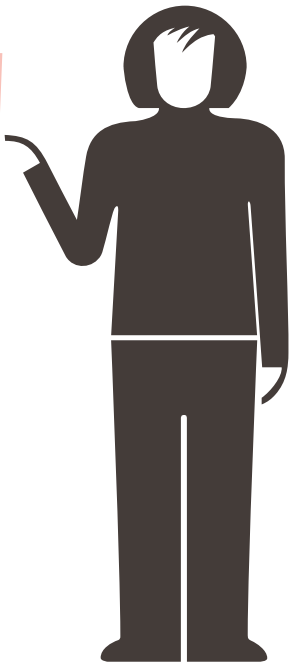
Kugler-Maag are a key player who aim to bring innovations to the automotive sector, including the use of agile software development methods and open source software.

Scaling Agile in Life Sciences

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

One more thing

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



Scaling Agile in Automotive

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

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

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

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

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

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

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

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

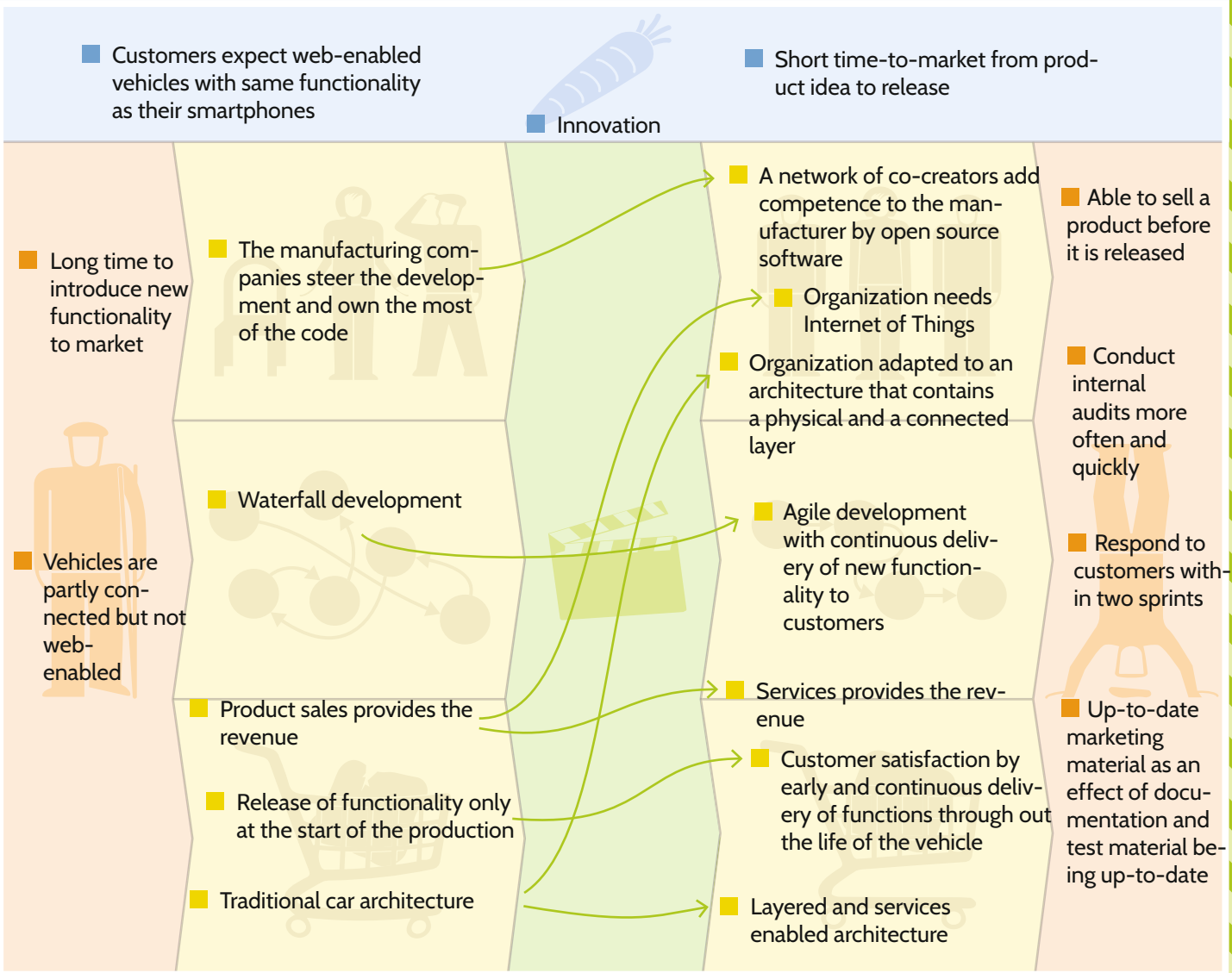
This is perhaps the most challenging part of the transformation, this that services become more important than products and only a small share of the profit is created through sales of physical products. The cultural challenges far outweigh the technological challenges.

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

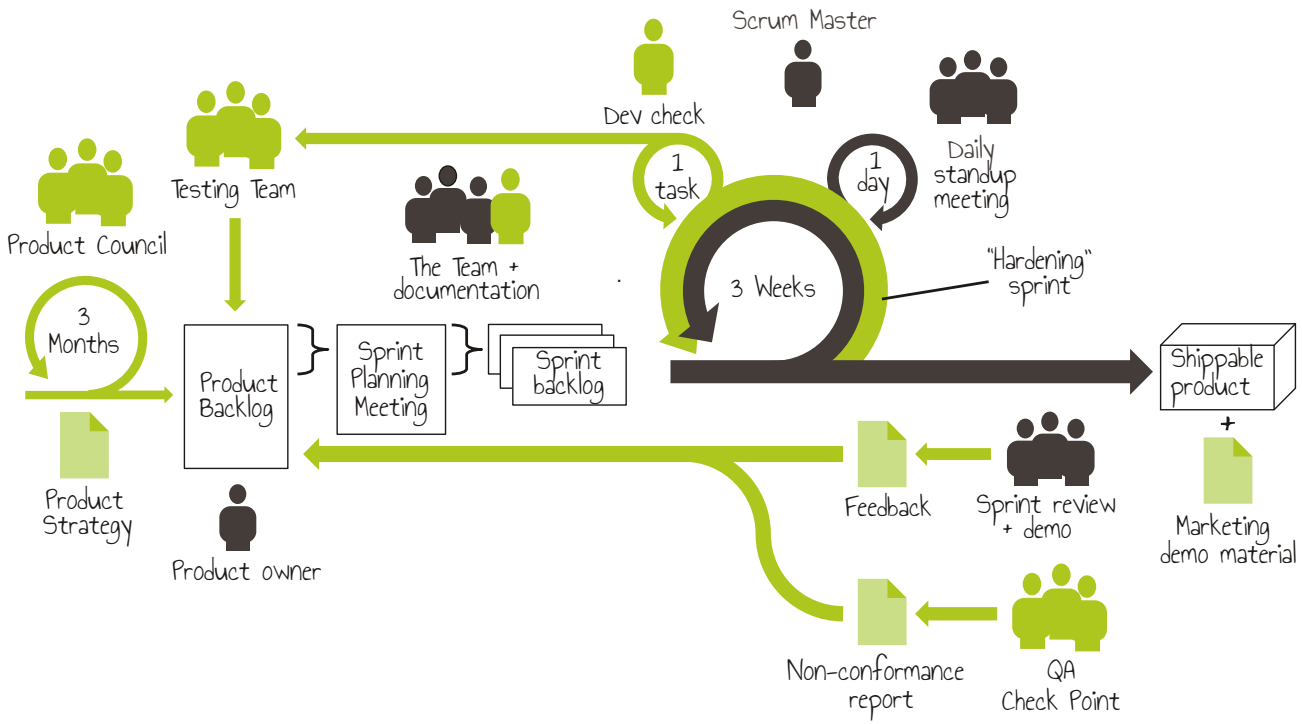
Once a car moves into its production phase, software development must carry on and add new functionality. Cars have to support updates and add-on apps that are developed after delivery. Naturally, the start of production-focused development has to be replaced by continuous development with short release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

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

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

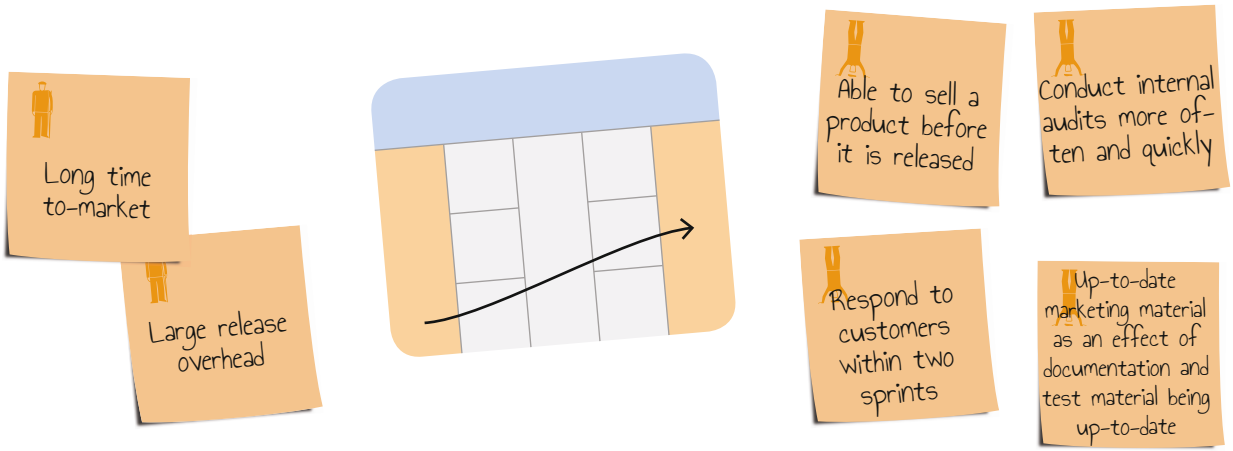


Scaling Agile in Life sciences



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

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



The Agile Manifesto offers four value propositions:

Individuals and interactions over **Processes and tools**

Working software over **Comprehensive documentation**

Customer collaboration over **Contract negotiation**

Responding to change over **Following a plan**

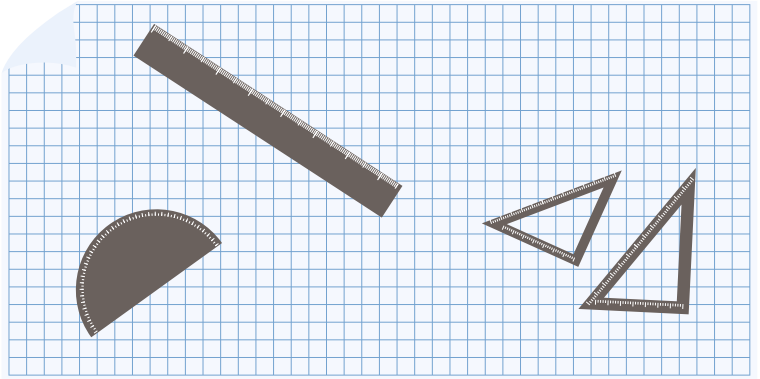
While agile advocates accept that the blue statements on the right are important, they value the red statements on the left more. However, in regulated environments, the blue statements on the right loom very large and are perceived to be key. If it isn't documented, it isn't done is a frequent refrain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

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

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

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

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



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

Lessons Learned

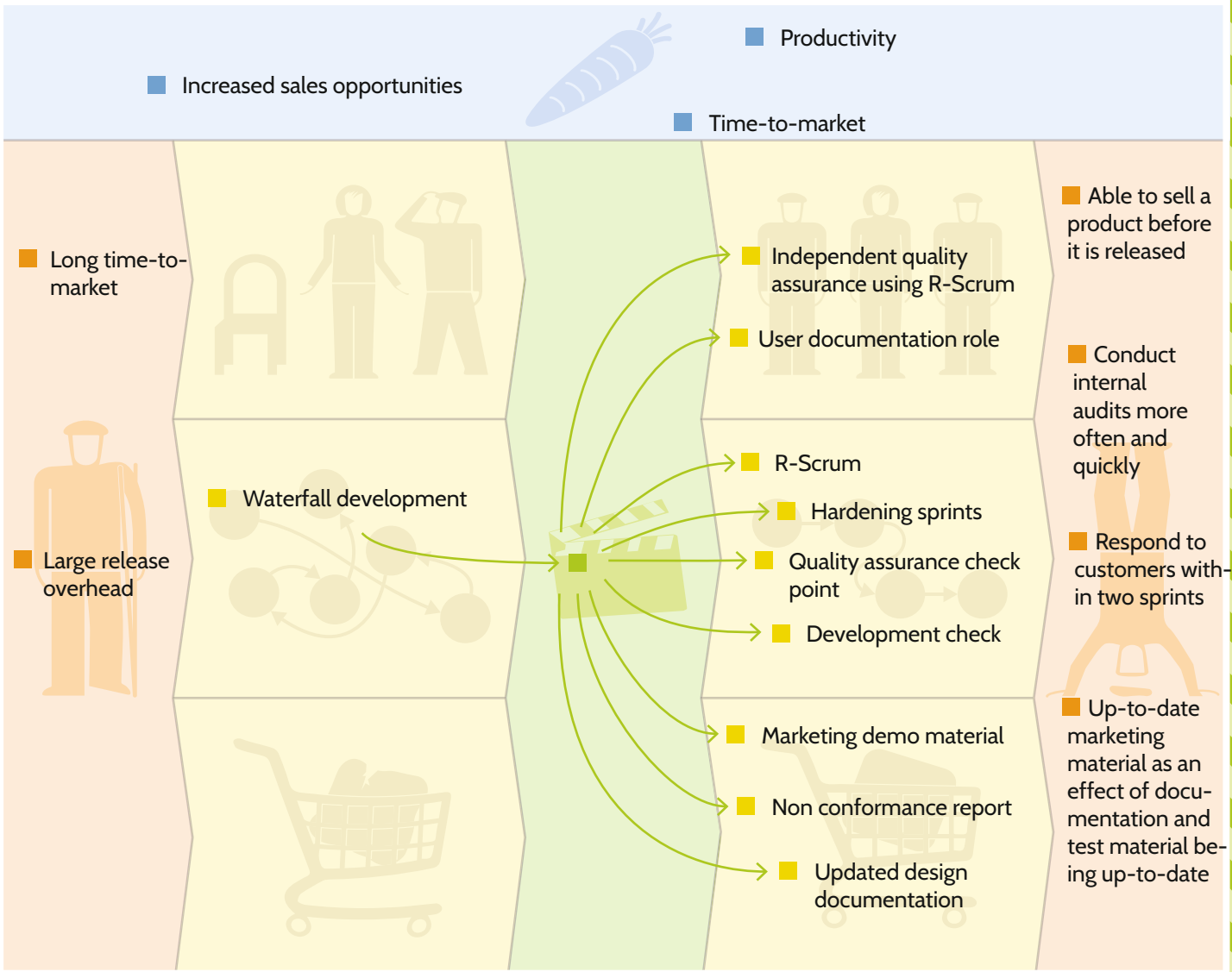
QUMAS have successfully tailored the standard Scrum framework to facilitate the additional constraints imposed by the regulations that their development process must consider. The key lessons learned of this case study are:

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

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

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

.



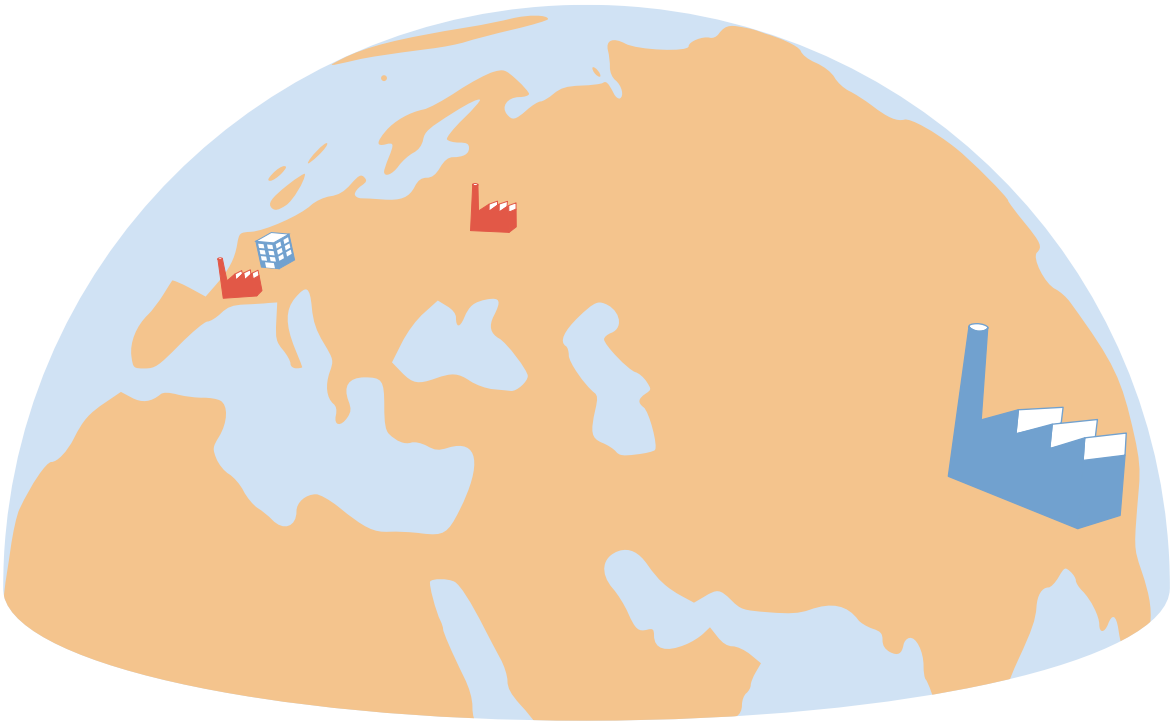
Outside the box

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

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

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

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



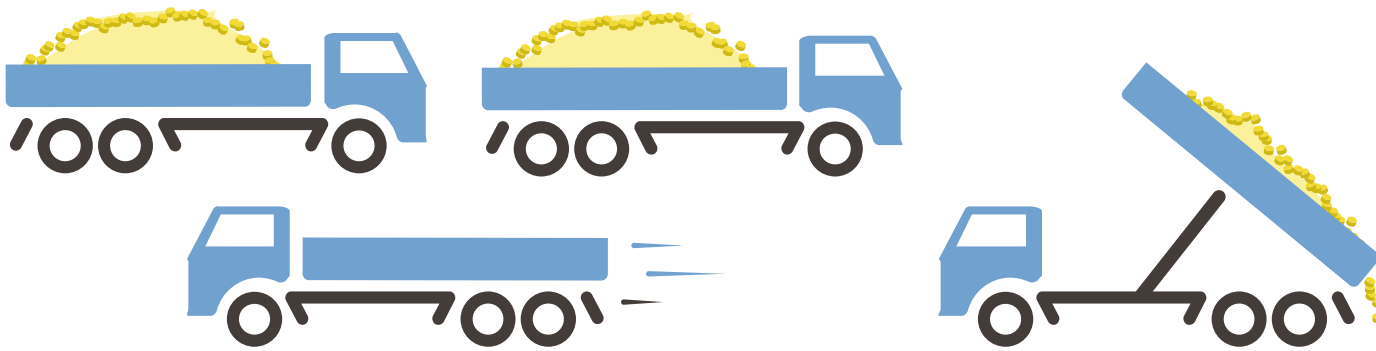
Putting the low-cost aspect aside, managing workload peaks is a good reason to choose outsourcing, as it is costly and risky to build up and maintain internal overcapacity. Further, to balance our investments and risks, it also makes sense to have our own developers working on the most important areas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

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

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

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

The unforeseen costs



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

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

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

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

We can expect additional costs due to:

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

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

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

We're only human

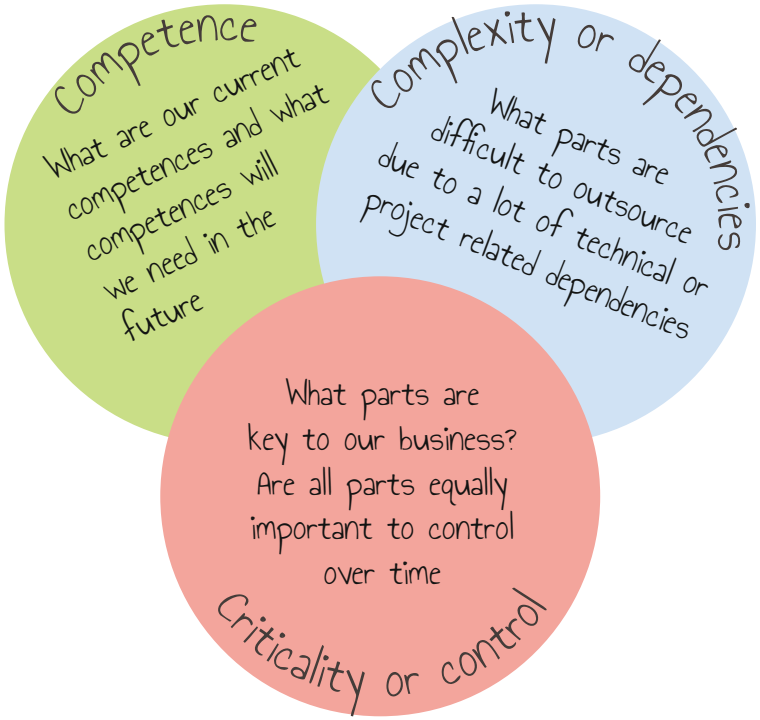
There will be challenges. Certainly, there are matters we can't predict and matters that are totally out of our control. But many problems that arise in an outsourcing or offshoring project can be traced down to human nature. It's due to how we communicate, how we motivate and are motivated, and make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

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

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

Due diligence

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



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

Our strategy

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

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

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

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

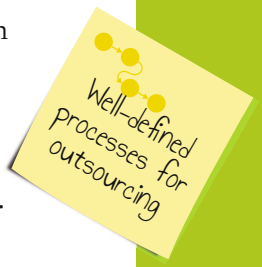
What do we want to outsource?

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

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

Who can we outsource to?

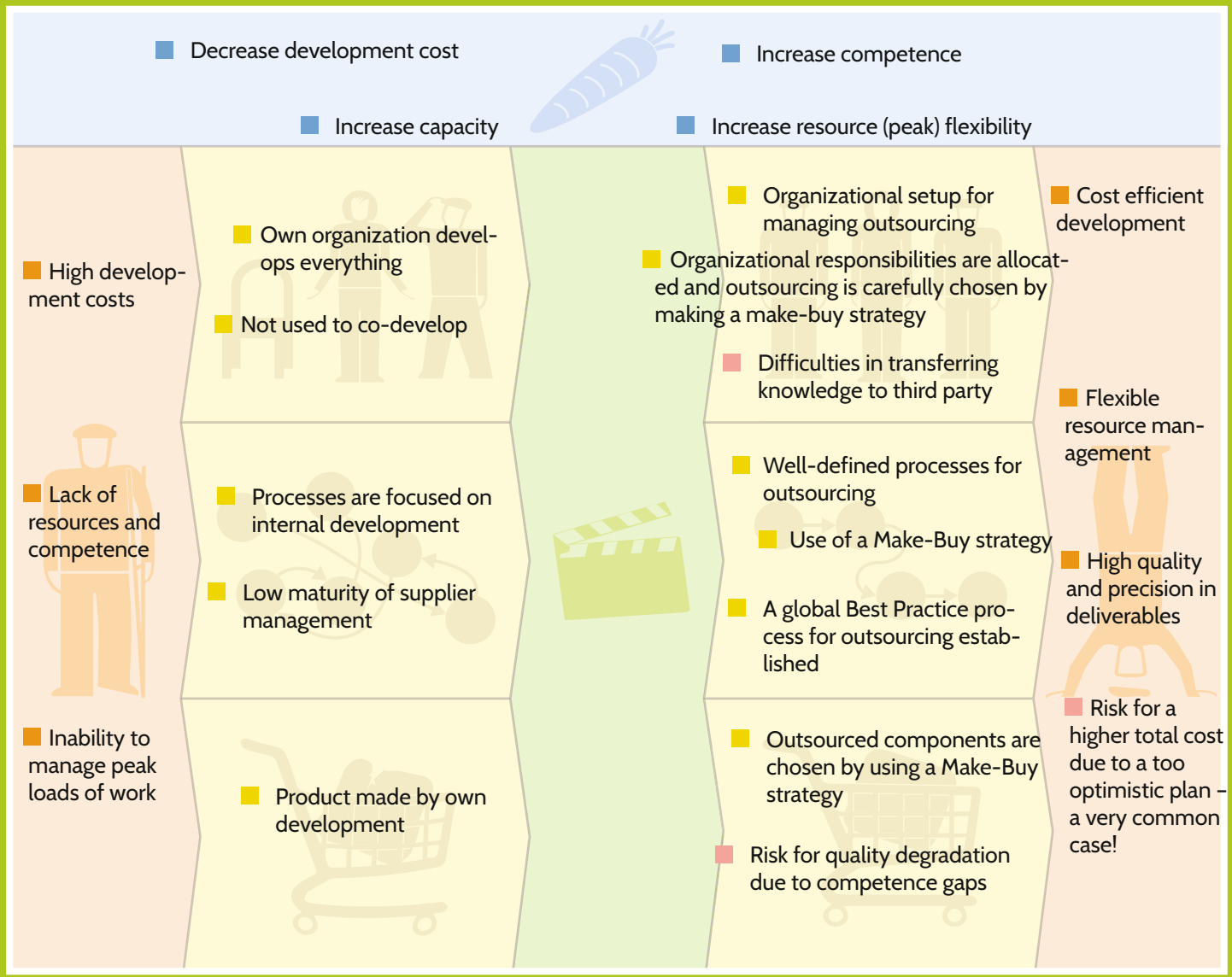
In addition to cost, we need to analyze the third party supplier's competence and maturity, development method and delivery model, their financial, legal and security status. We need to understand the political situation, and geographical and cultural contexts. If our development team will be tied up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



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

How can we organize ourselves?

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


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

Get inspired

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

Efficient communication in a global delivery model

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

Outsourcing Strategy at Sony Mobile

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

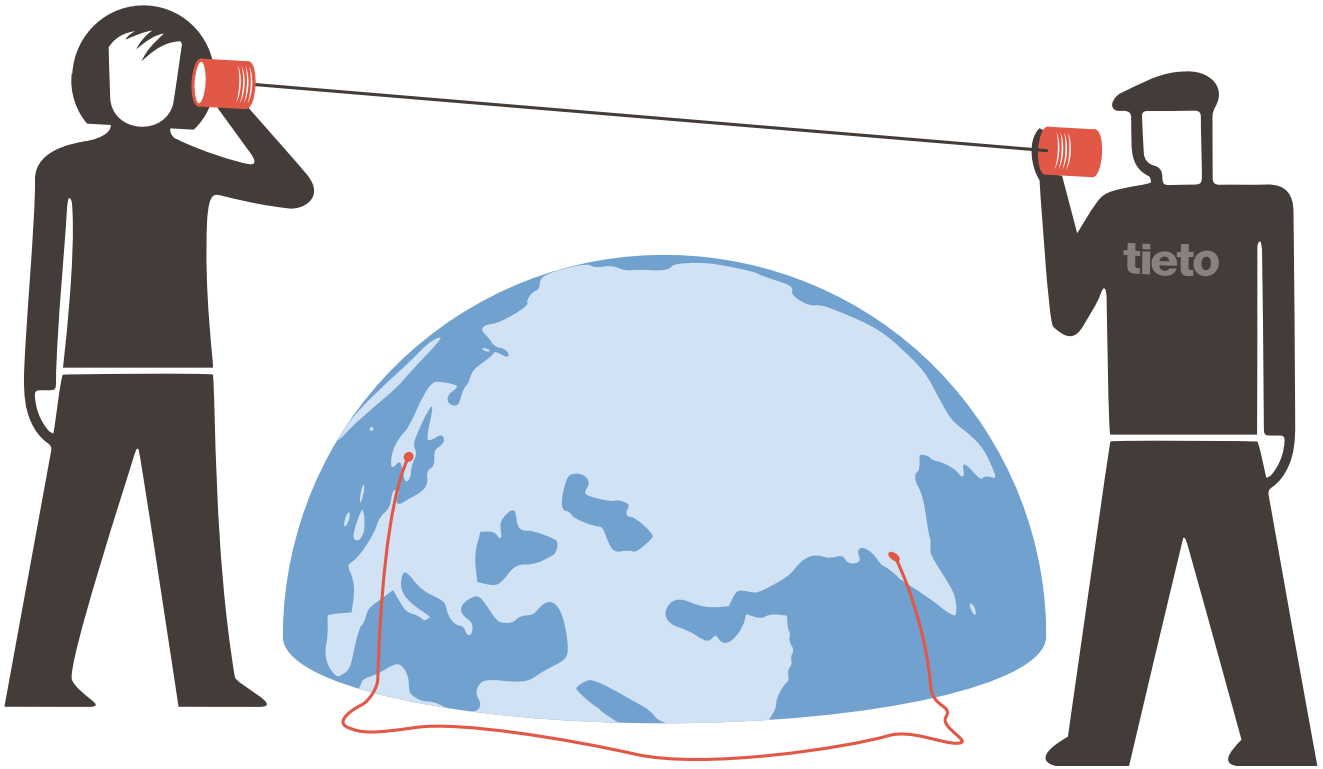
Not so shore anymore

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

Play it again, Sam, backwards

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

Efficient communication

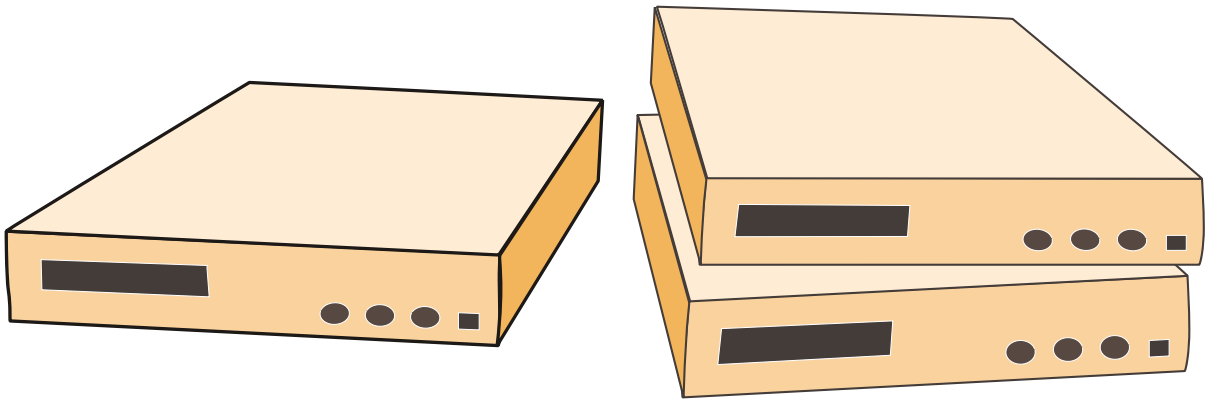


in a global delivery model

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

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

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



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

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

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

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

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

Observations

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

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

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

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

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

Recommendation on how to succeed

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

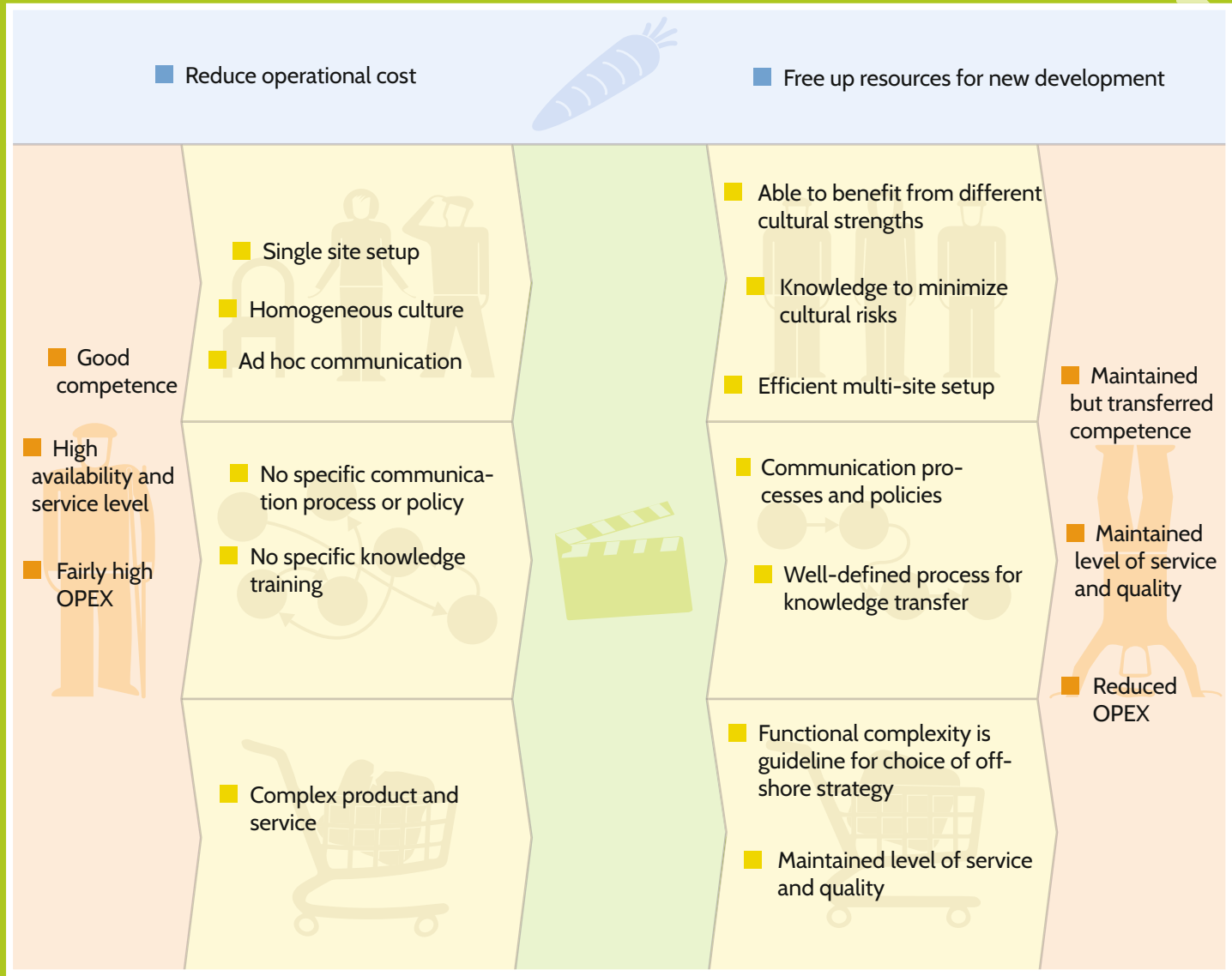
Decide upon a competence strategy early in the offshoring phase. Take this into account when staff reduction starts at the local site. It's important to secure personnel for future roles available in later phases of offshoring.

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

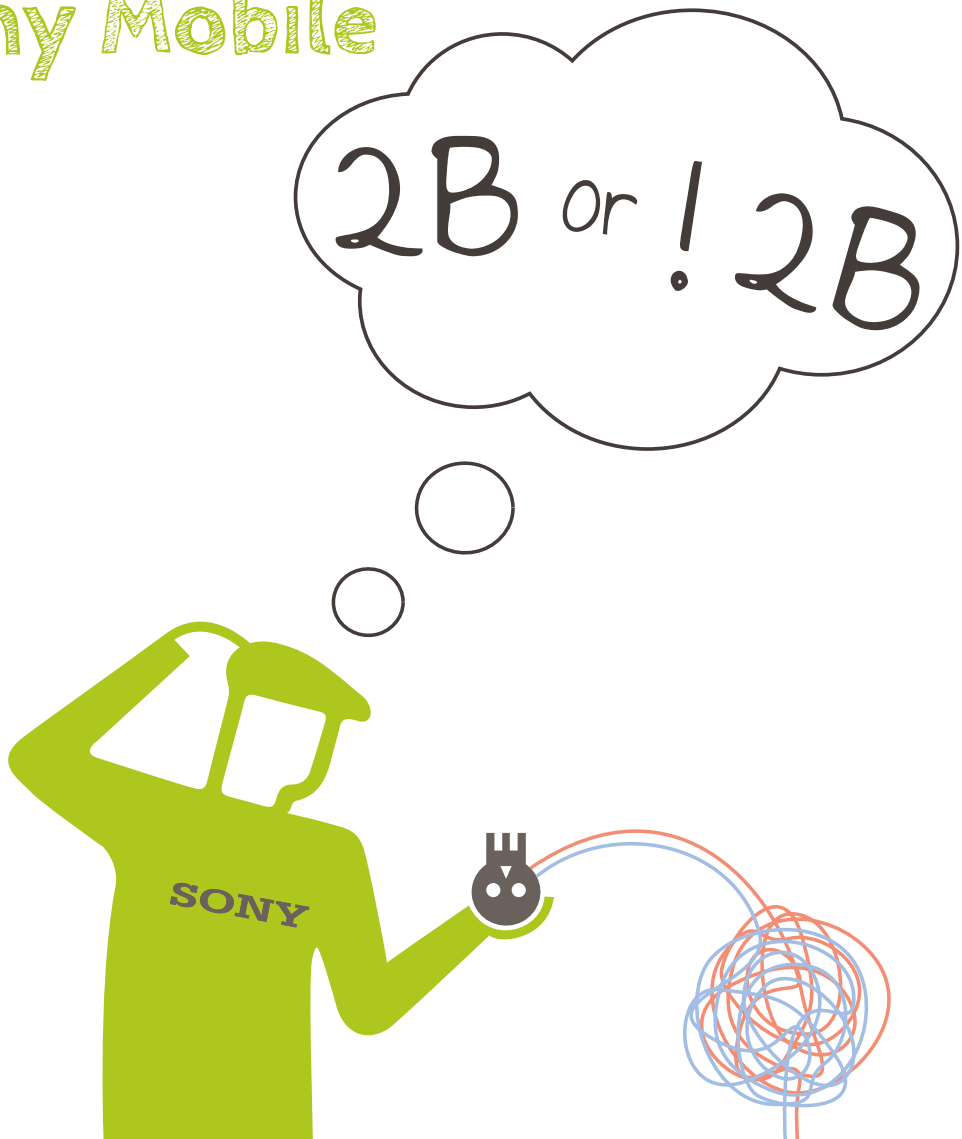
1) Self-study using existing training material (docs and videos)
2) Supervised trial operative work
3) On-site training with architects
4) Unsupervised operative work

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

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



Outsourcing Strategy at Sony Mobile

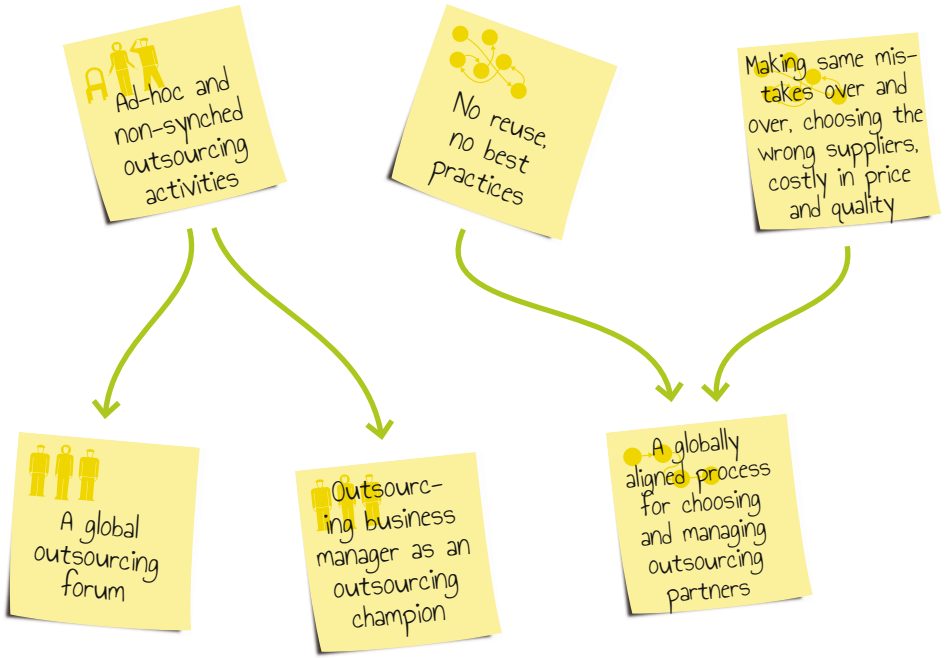


Software has transformed the mobile phone industry. Whereas the first mobile phones contained a minimal amount of software, today mobile phones contain more powerful processors than those used to put man on the moon. This allows modern phones to do much more than just making phone calls, offering many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

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

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

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



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

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

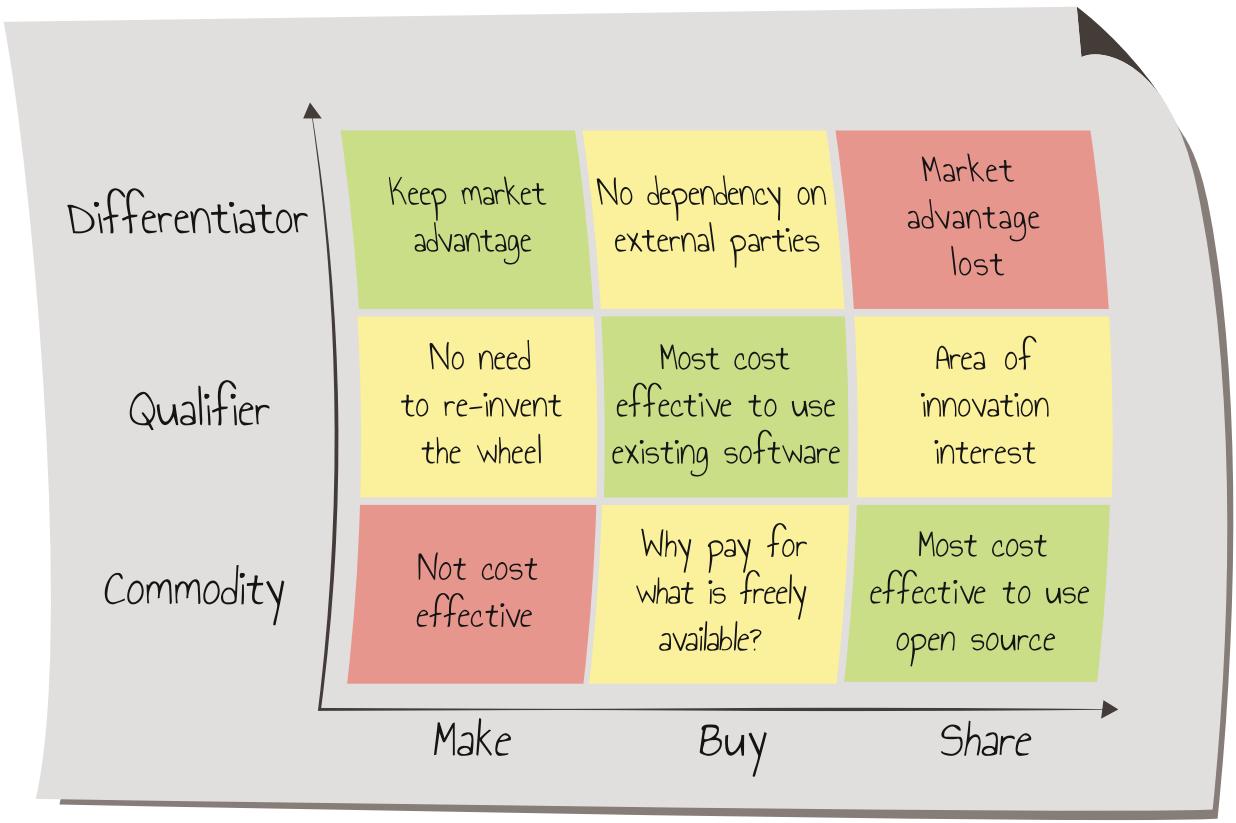
The second activity was to create a decision framework to help business units analyze and select suitable components to outsource. Using a tool support, business units can evaluate components based on a set of three key parameters.

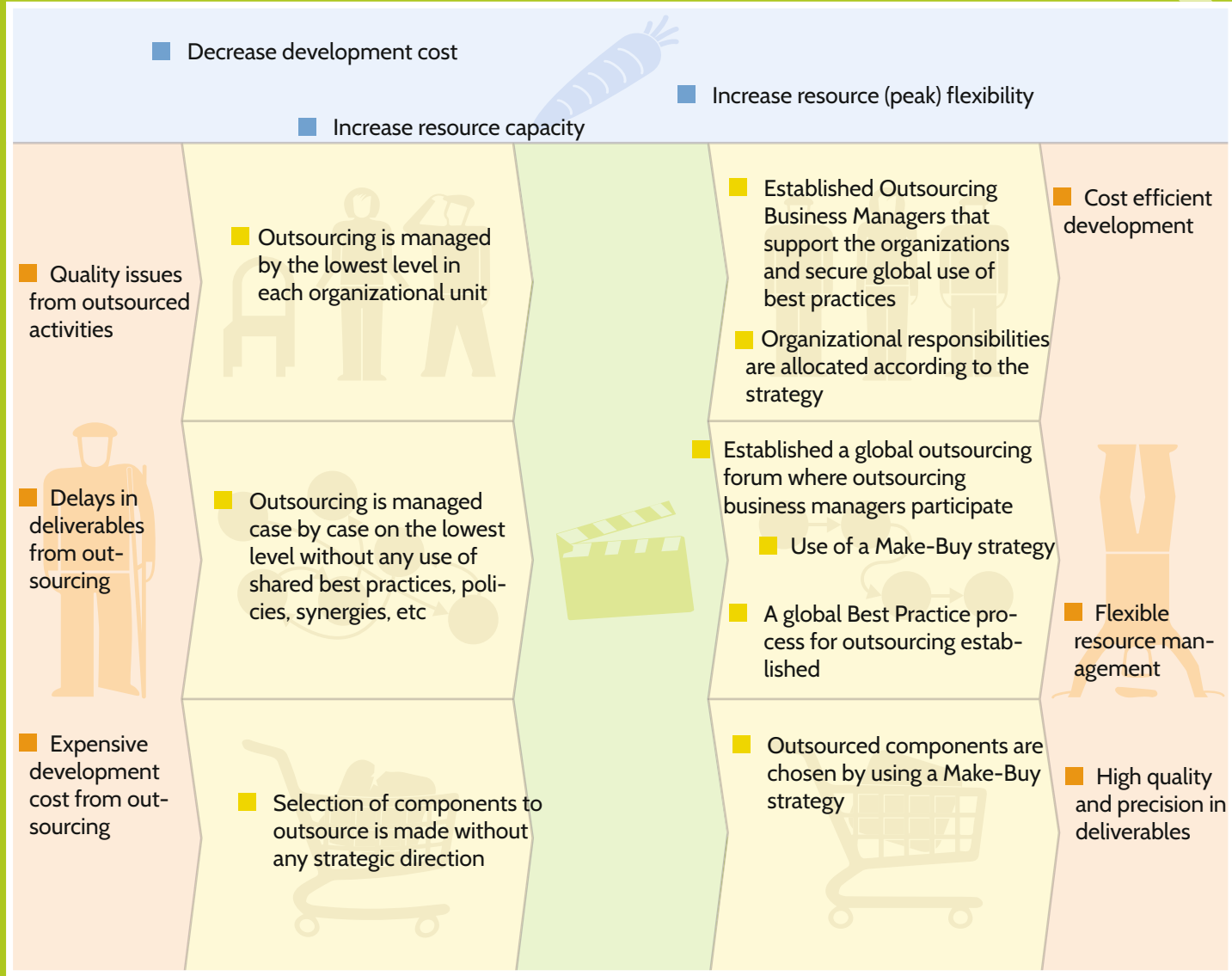
The first parameter refers to **current capabilities**, which refers to competence and amount of resources. What is the current capability for a given component? Is there a lack of competence to implement or maintain the component? Are resources wasted on components that can easily be acquired from a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

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

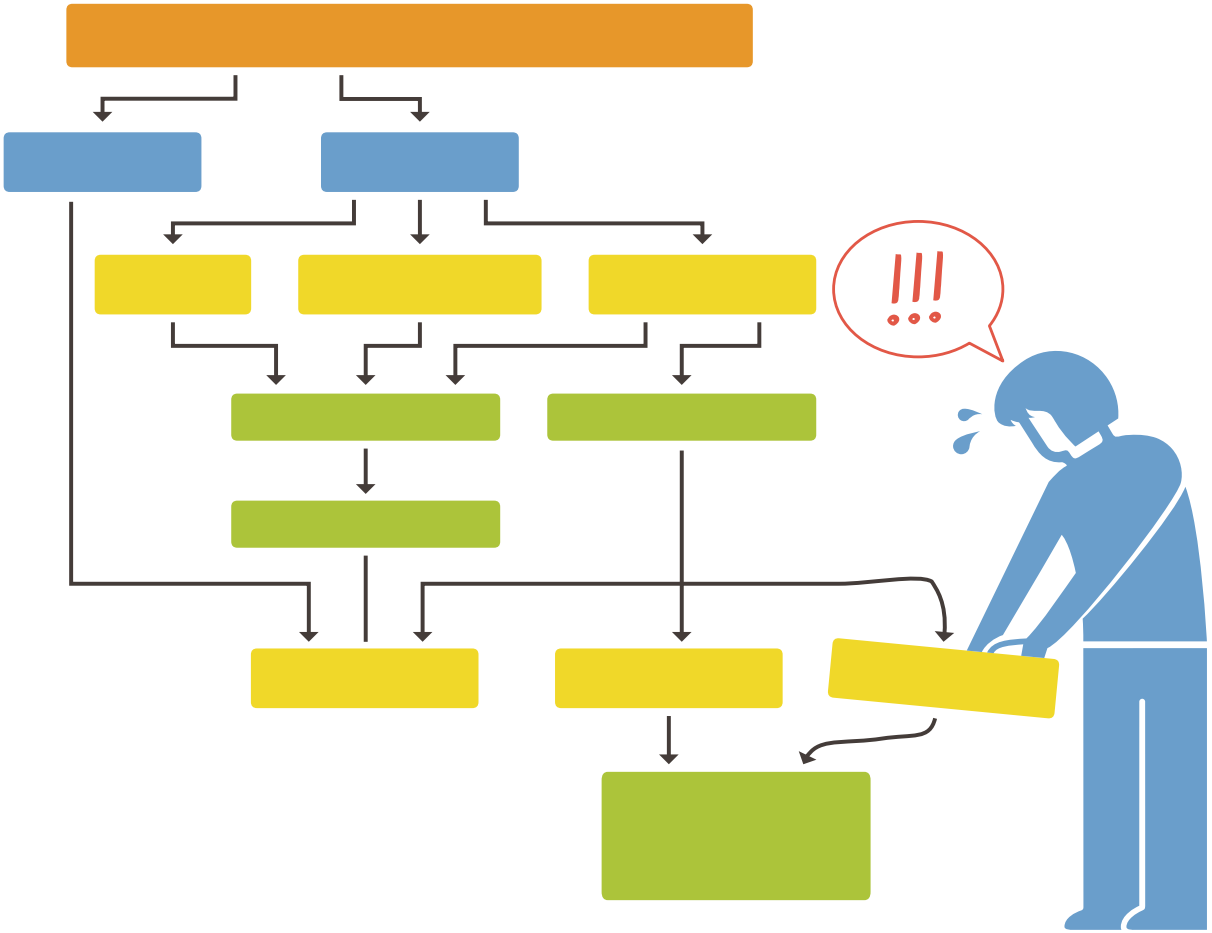
The third parameter is concerned about **long term control and competence**. This is an indicator of whether or not it is important to be able to control a given component's roadmap and future evolution. When outsourcing (or opensourcing) a component, a certain level of control is lost. Components that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

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


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



Not so shore anymore



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

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

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

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

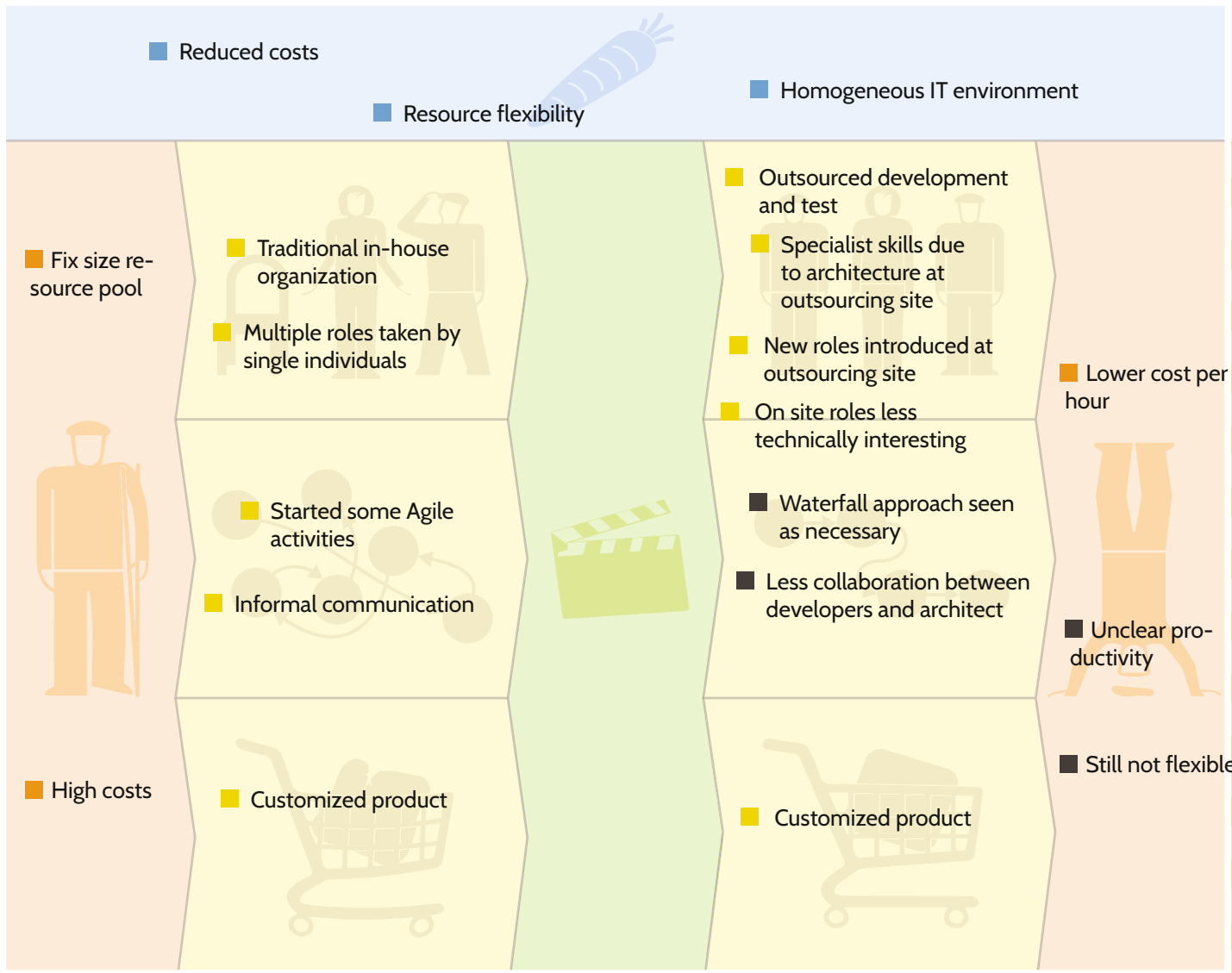
At the outsourcing organization there were developers as in the original organization, but also team leaders responsible for leading the work and keeping the contact with the original organization. Turn-around times for development became longer and it became much more difficult than anticipated to find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

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

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

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

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


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

Play it again, Sam, backwards

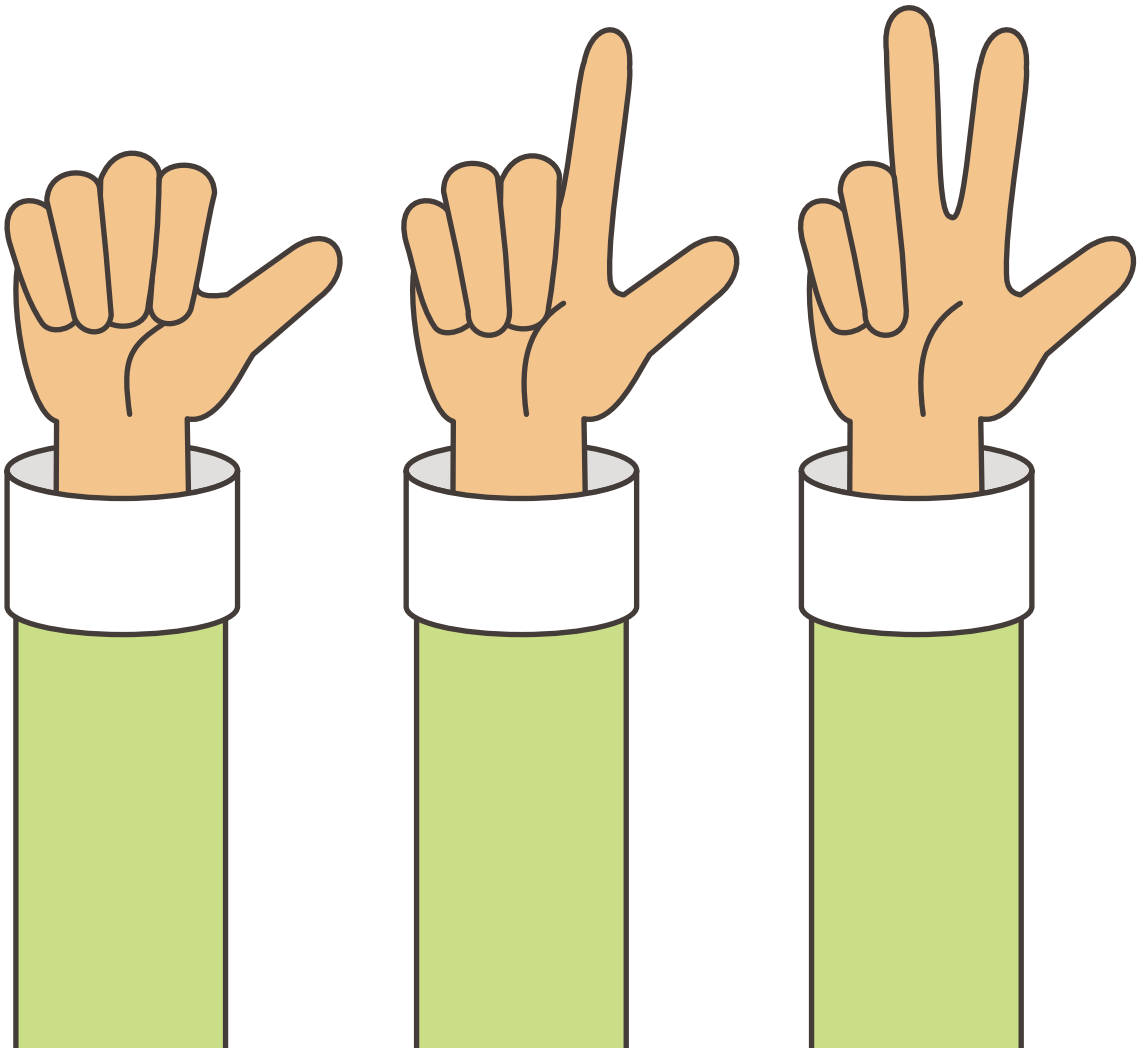
PlayNow was Sony Ericsson's download service for media such as music, games, ringtones, wallpapers and themes. They used to have the bigger part of the PlayNow team outsourced. The development was taken care of by the outsourcing partner and Sony Ericsson took care of the project and product management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

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

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

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

First things first



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

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

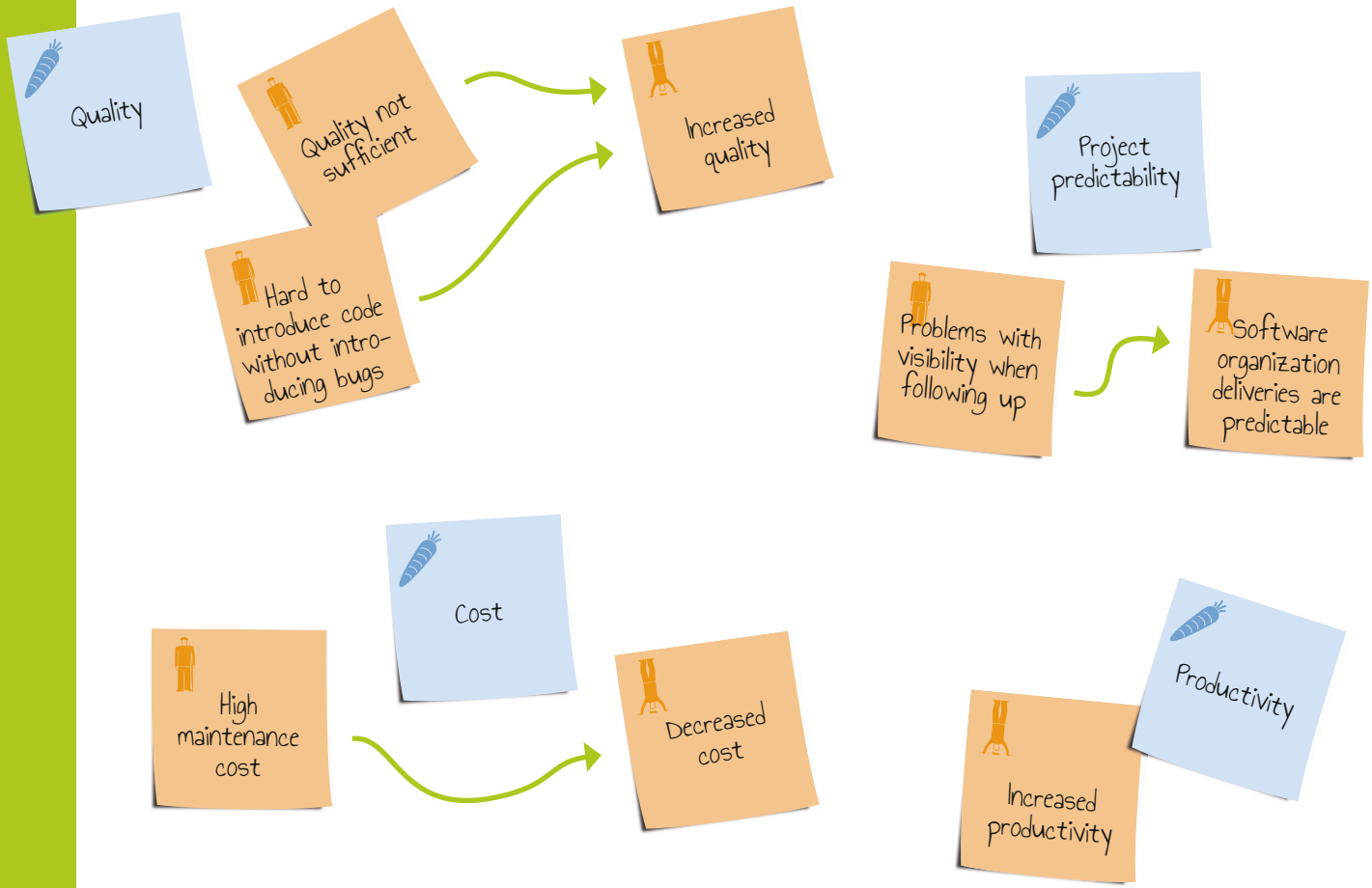
The first things are about the very basic needs. We need for instance to know what to do and how, when and where to do it. We also need to check the quality of it.

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

A good read

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

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

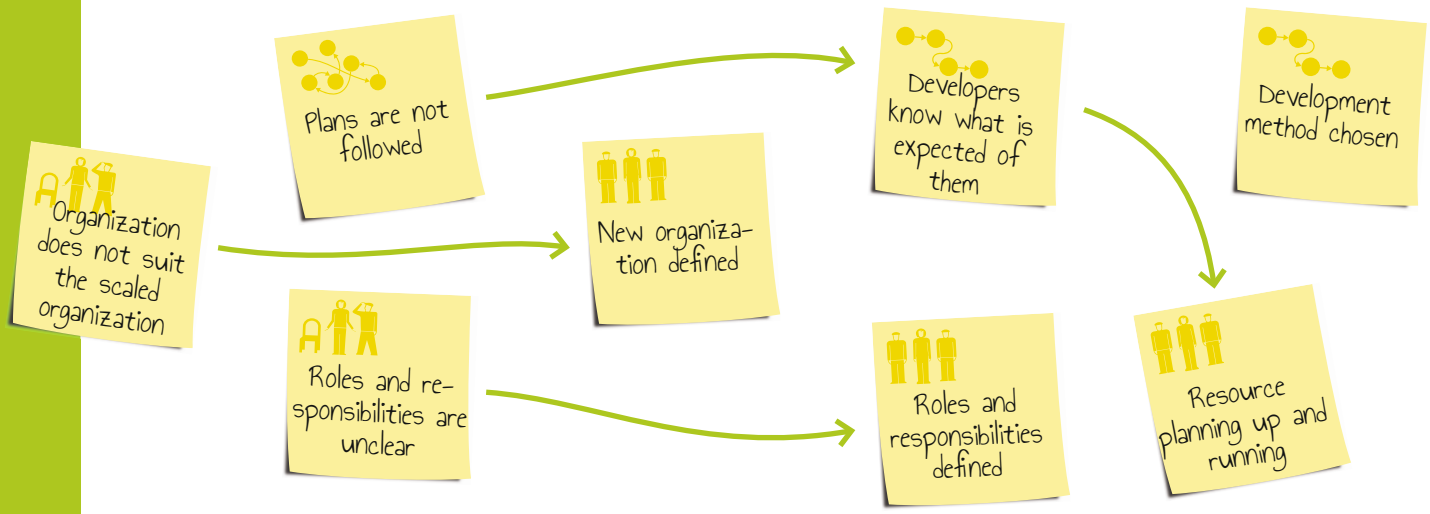


One of the reasons why we have these problems is probably that our organization has outgrown our current way of working. Redefining the way of working to match the current size of our organization will in general lead to increased quality, productivity and efficiency of the business. It will in particular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

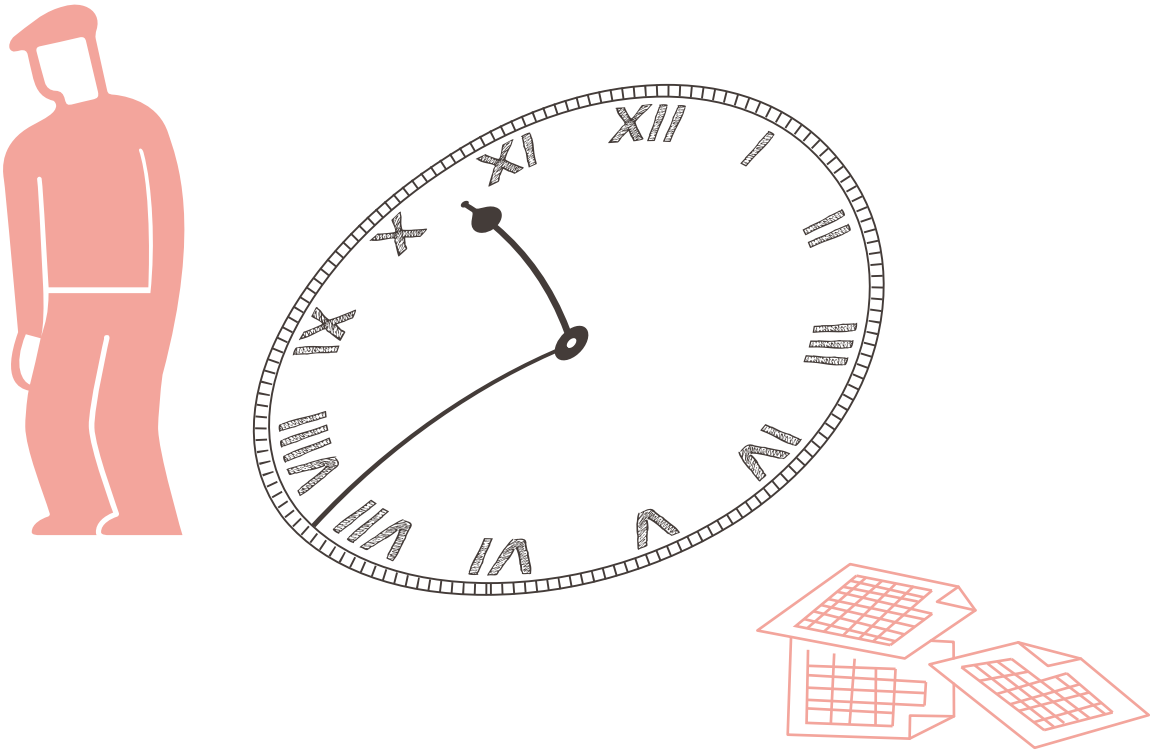
It's hardly a surprise that growing organizations get problems. To do a good job, basically developers need to know "what to do" and "how, when and where to do it". As easy as one, two three, one might think. As a matter of fact, it gets more and more difficult to spread information as the organization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

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

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



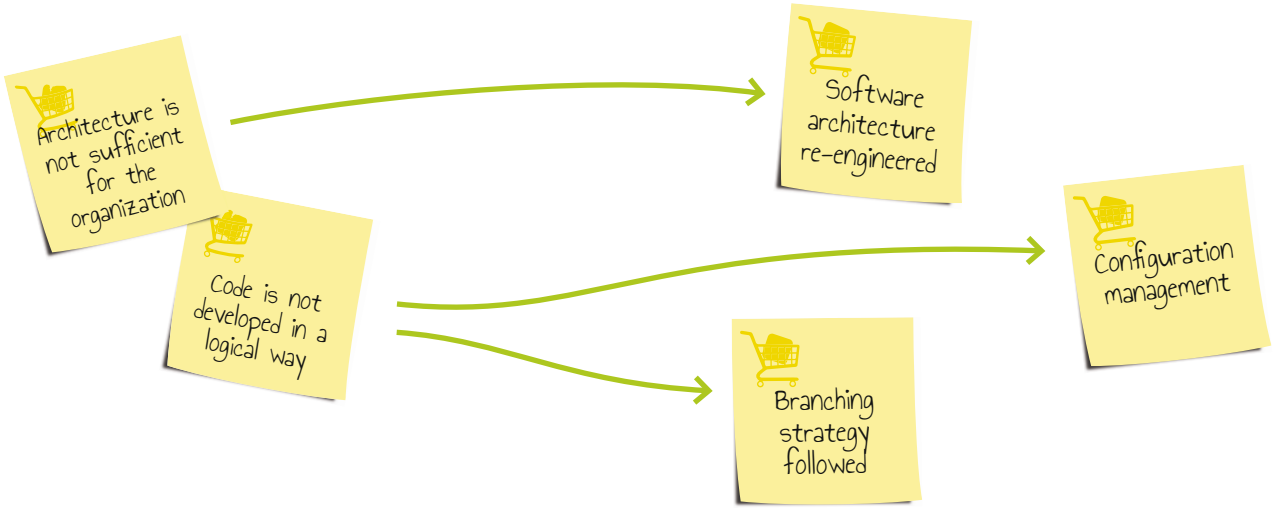
All engineers need to know their roles and responsibilities. The organizational structure must facilitate effective communication in all directions, not just vertically. The optimal structure closely follows the ways people work, weather they work in projects or in a product line. Agile development methodology,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



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

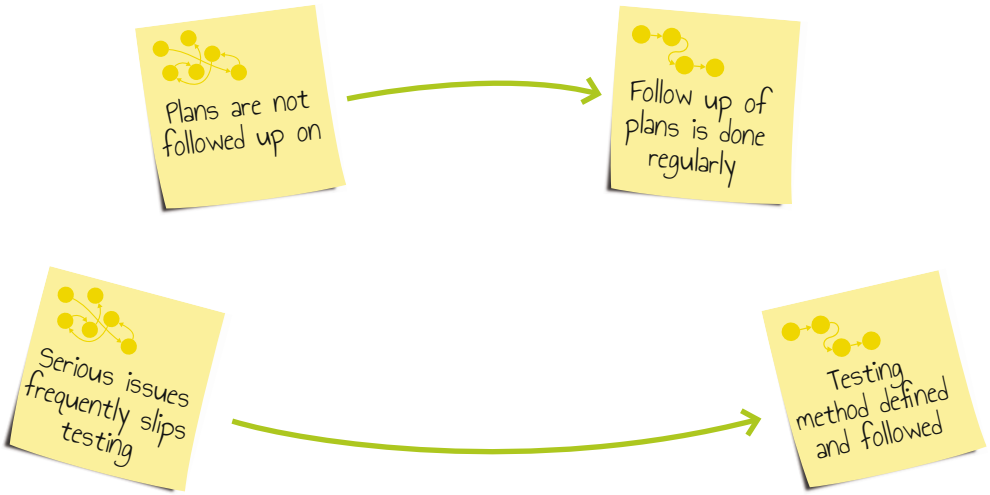
In what state is our software architecture and how do we manage it? The software has to serve its purpose, to fulfill the function for which it was created. It must also be firm in terms of structural integrity and durability. If optimally designed we should be able to:

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

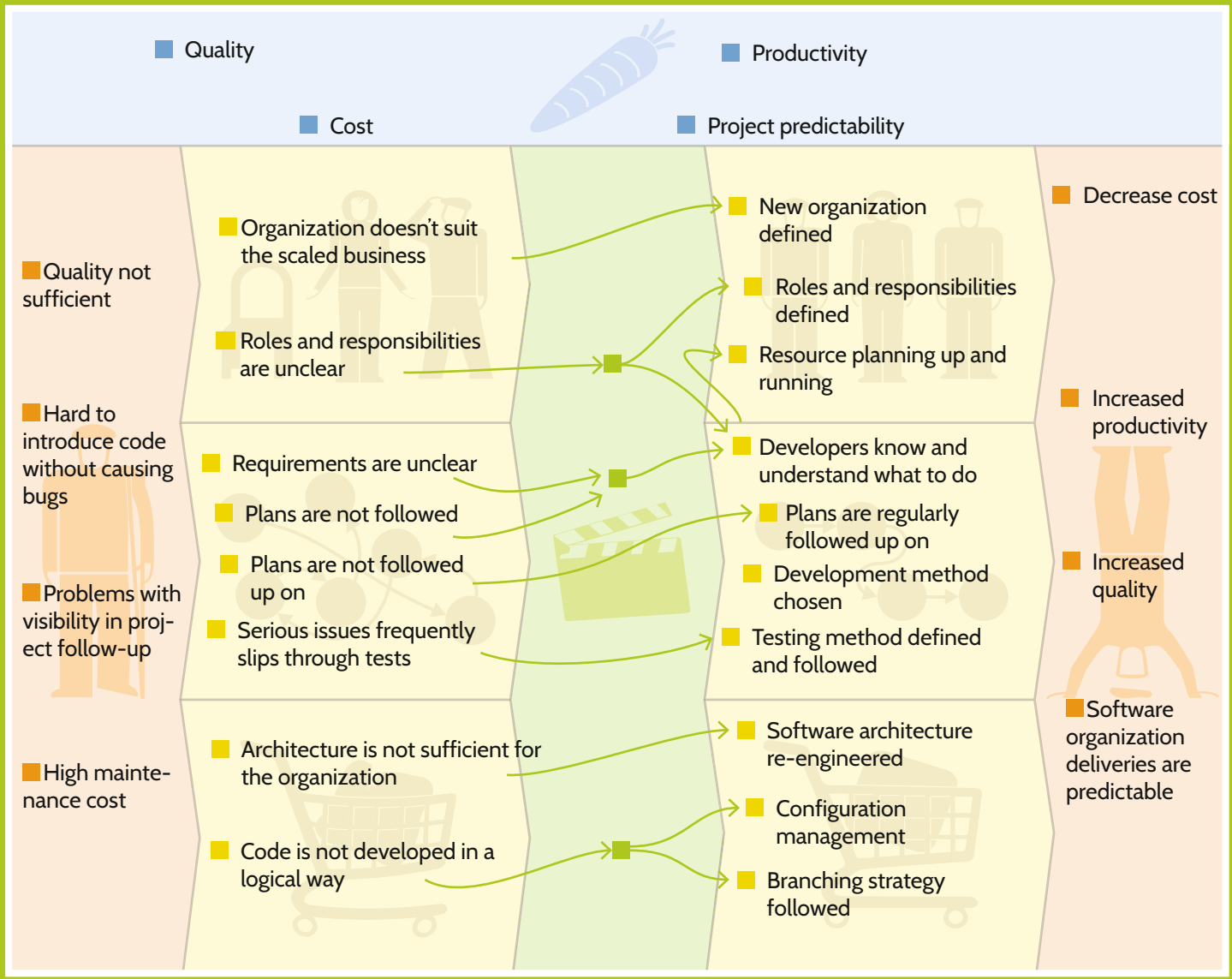


It's key to design the system as a set of clear-cut parts embodying well-defined boundaries. Software architecture is, simply put, how these parts are structured and how they relate and communicate with one another. The architecture starts with its documentation, a blueprint that governs how to design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

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



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


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

Get inspired

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

Robotic growing pains

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

Softhouse reflects on architecture changes

Learn about the effects of architectural changes in Android development.

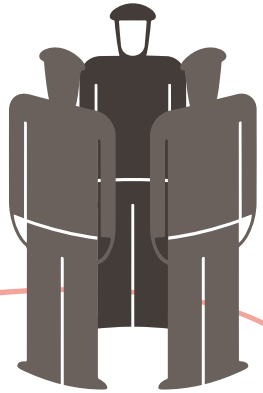
From mobile to platform

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

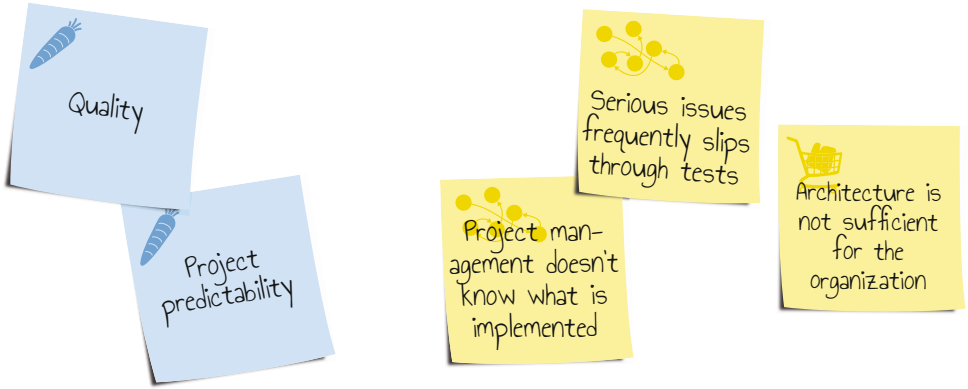
One more thing

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

Robotic growing pains



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



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

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

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

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

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

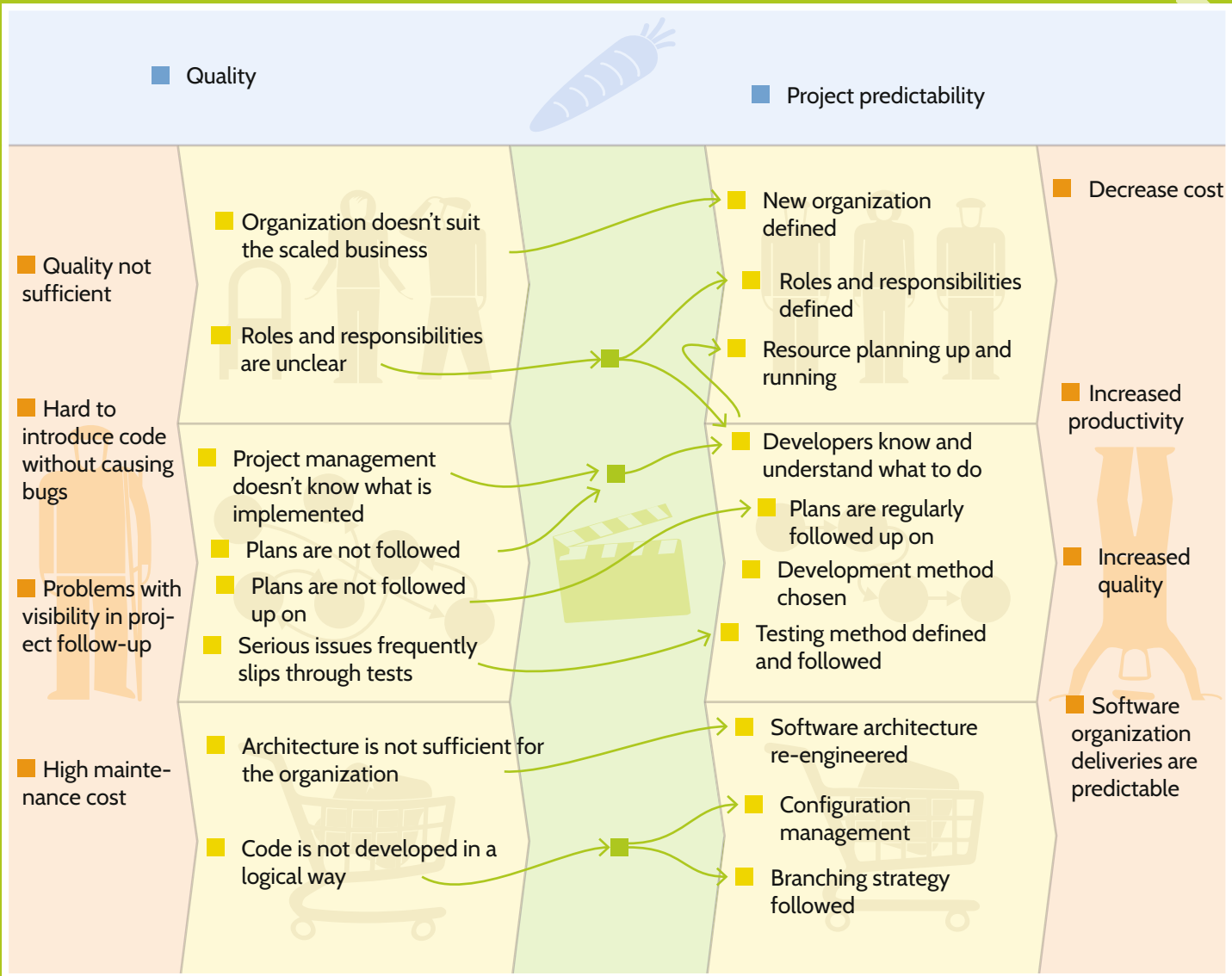
A new meeting was held to agree on which changes to prioritize. The software organization, project management and product management were all involved in this decision.

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

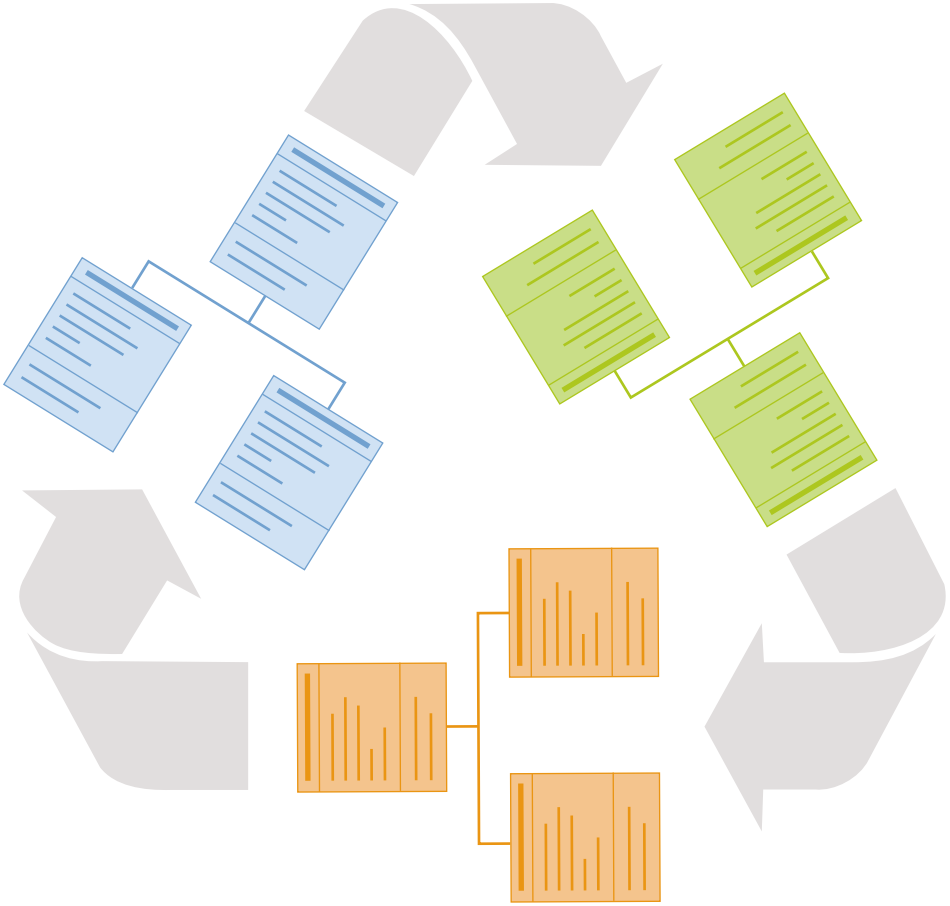
In April 2016 the software organization had a completely new situation. Instead of the negative atmosphere that characterized 2014, a more positive attitude characterized the organization. The software team was positive to the new ways of working and they believed they were working in a good way.

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

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



Softhouse reflects on architecture changes



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

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

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

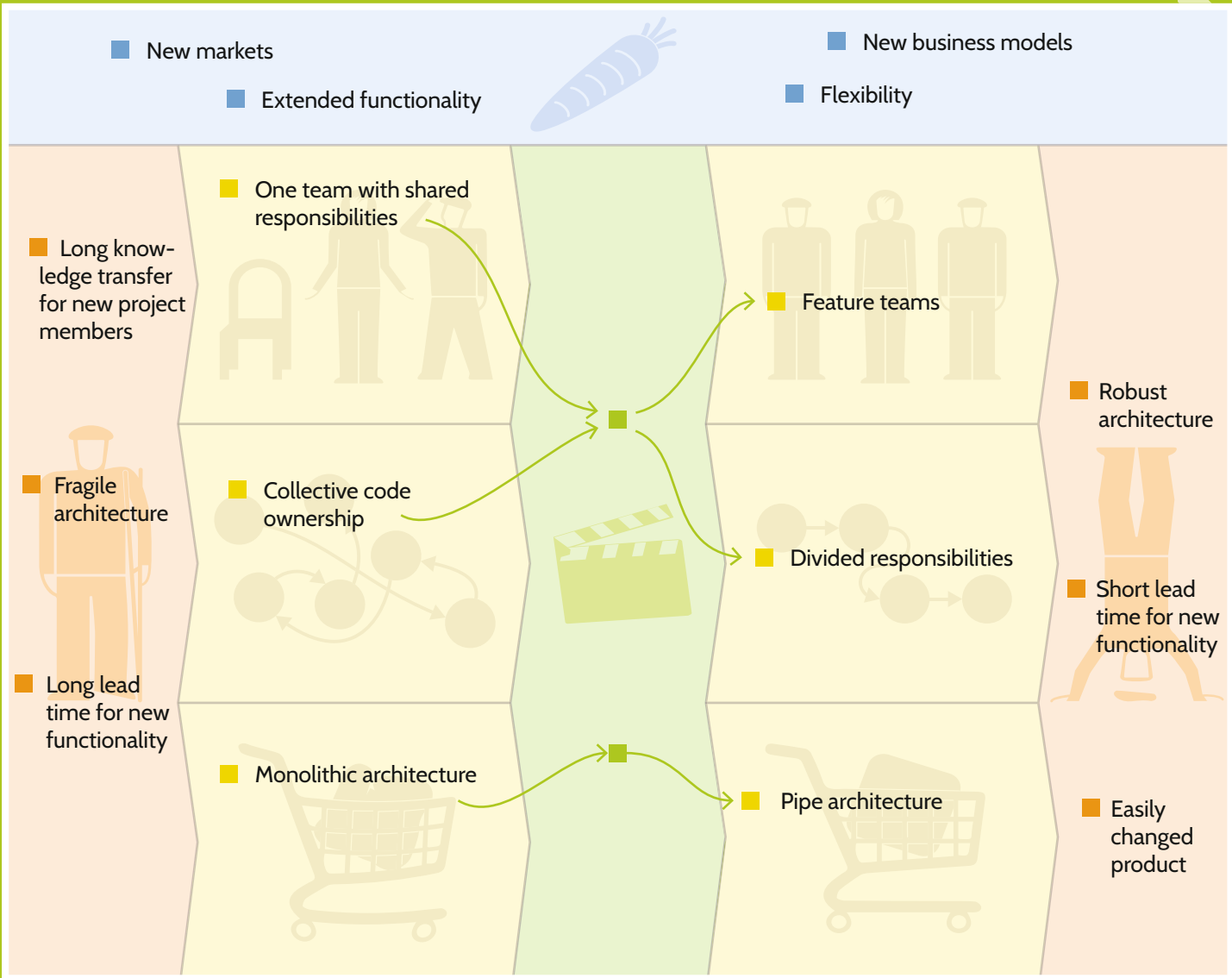
Starting as a small prototype, they created a client system to provide their customers with data from their internal information systems. As the usage of the client system grew, the product itself and the number of functions increased rapidly. A problem was for instance that any changes in the product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

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

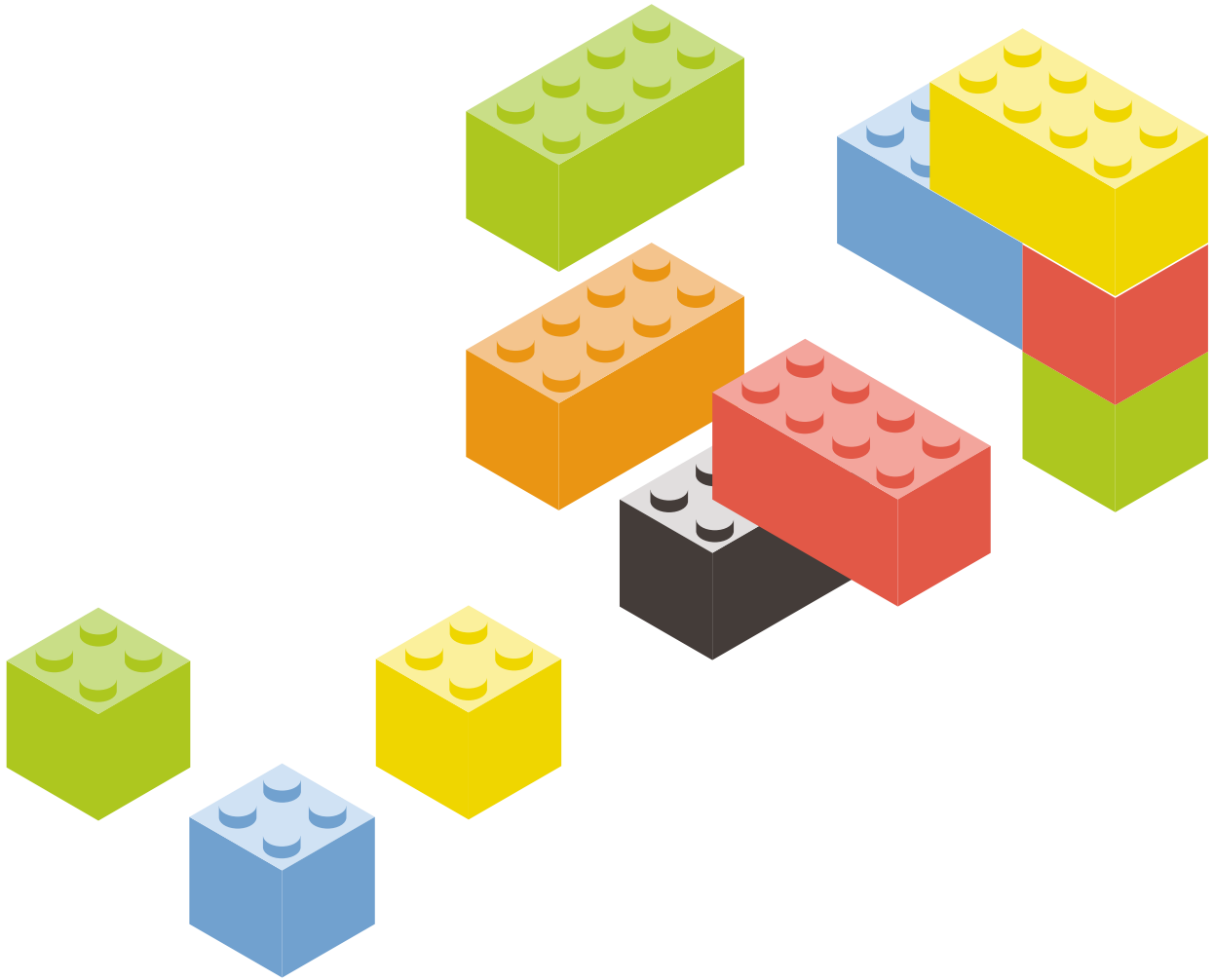
To make the design less fragile, the architecture was divided into modules. Each module implemented one specific function provided to the user. Functional decoupling allowed the developers to make corrections or updates of existing functions more efficiently and with higher quality. It also allowed for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

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

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


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

From mobile to platform



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

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

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

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

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

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

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

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

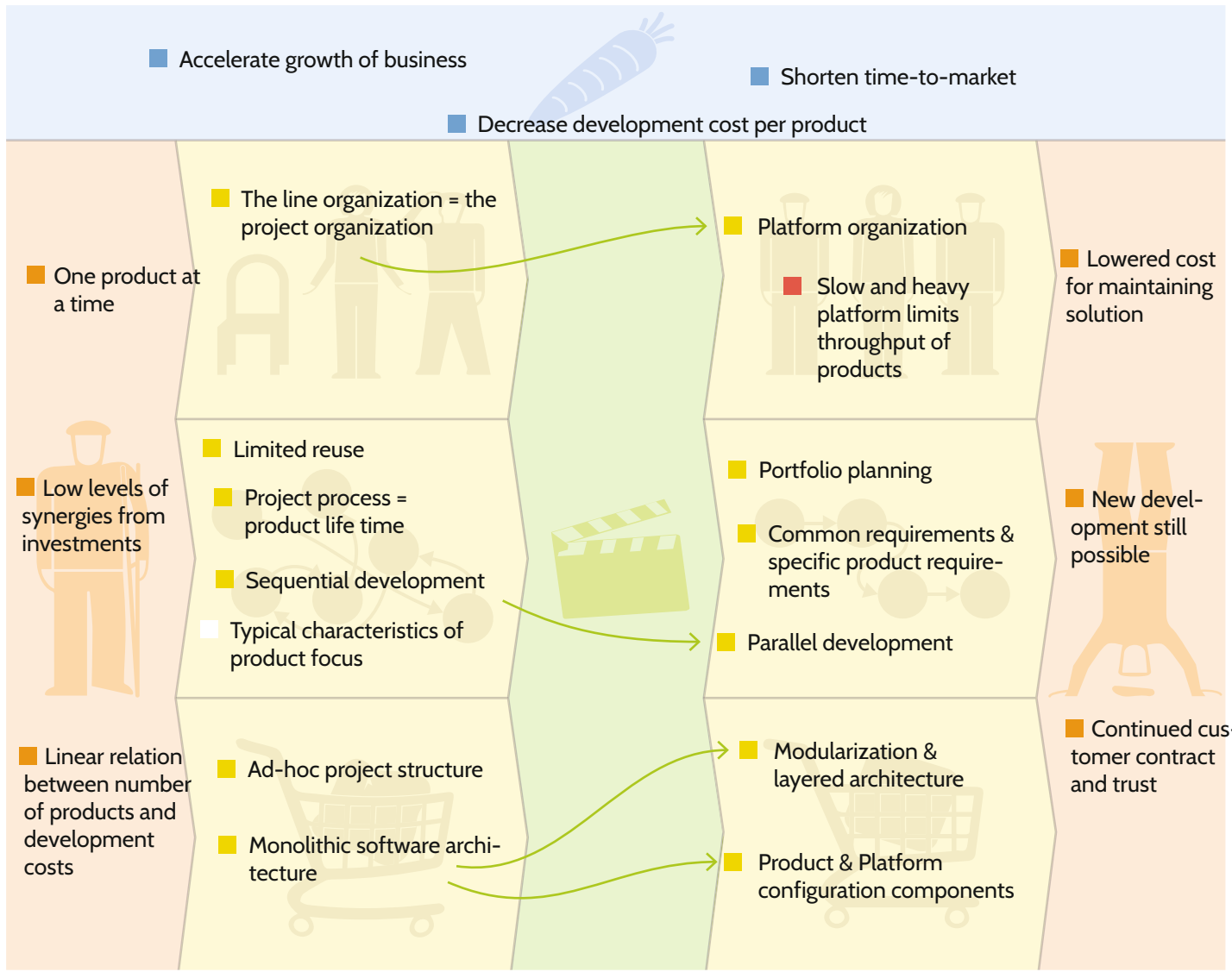
– So did they continue to improve and become better and better?

The answer is no.

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

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

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



Your

B. Fitzgerald et al., *Scaling a Software Business*,
DOI 10.1007/978-3-319-53116-8_3

journey

How you get there

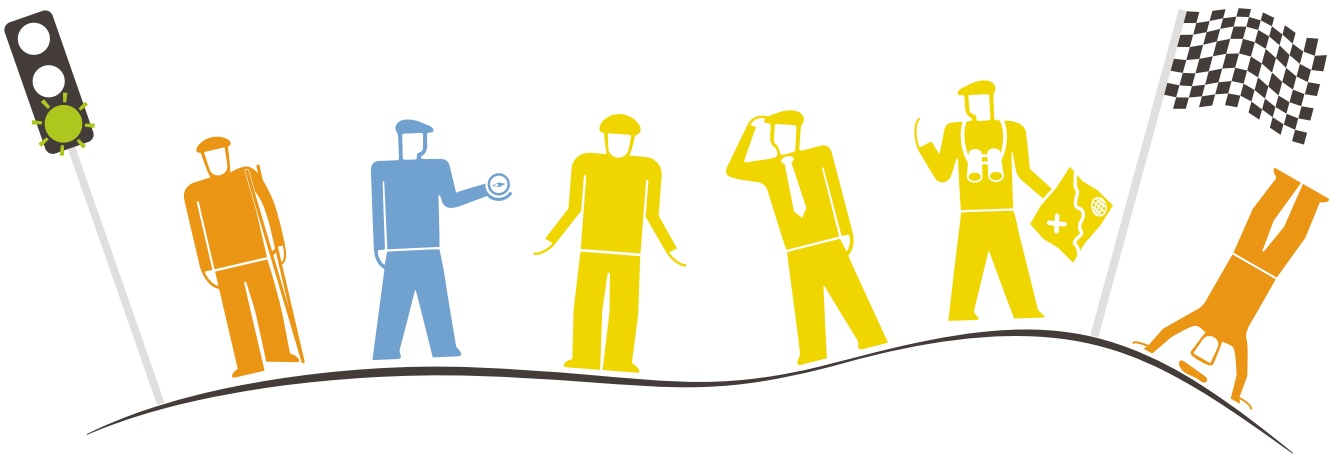
Good old post-its

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

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

The workshop in short:

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



Setting up the workshop

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

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

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

Some practicalities before we start the first workshop:

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

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

Let's start! Why are we here?

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

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

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

What is our strategy? - What are the drivers?

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

Start with the external drivers

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

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

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

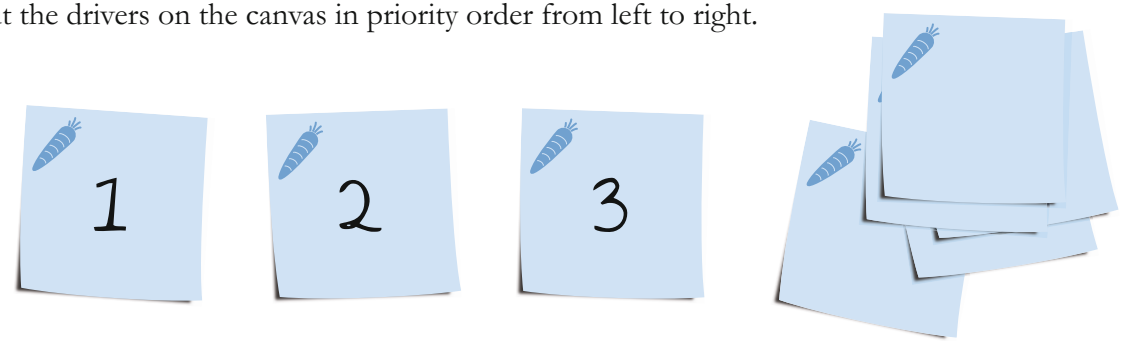
ny to a successful future. Is it the OPEX cost that needs to be lowered or is it rather a complete new business model that will outperform our competitors?

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

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

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

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



Good things we want to keep

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

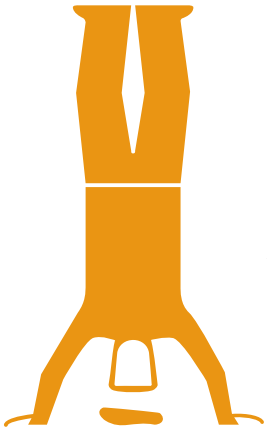
Continue with internal drivers

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

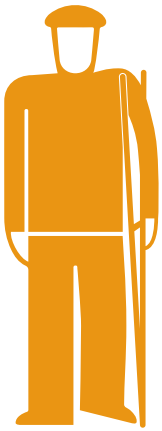
- What would make us proud?
- What kind of work climate do we want to have?
- What do we need to be well prepared for the future (which is sometimes not so easy to predict – so why not become as prepared as possible)?

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

Desired abilities and current inabilities



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



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

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

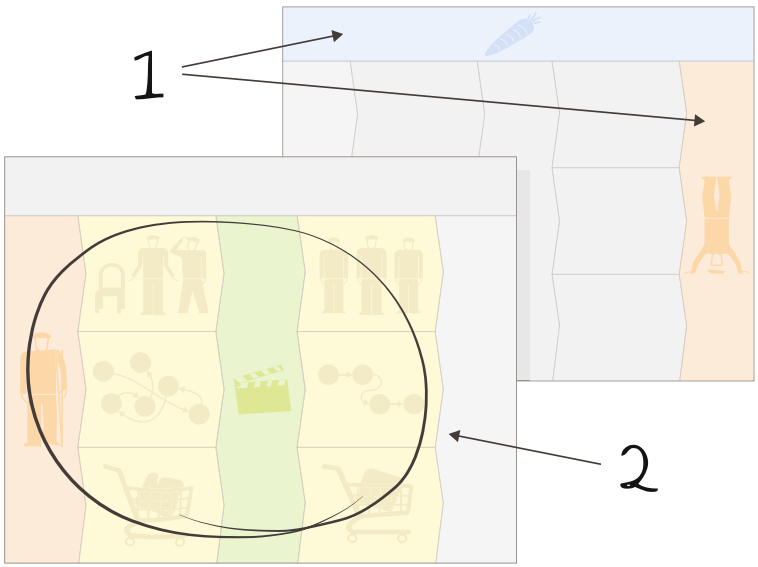
When the desired abilities are in place continue with the current inabilities. What is it that is not good enough – what are our growing pains? Obviously, many of them will be similar to the desired abilities, like for instance customer support availability: a desired ability is 24/7, but current inability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

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

Iterative process

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

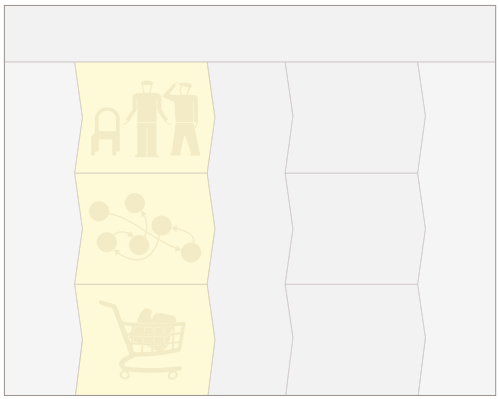
In many companies, the people that define drivers and abilities, and the people that can break these down in terms of organization, process and product, are not the same. In such cases, it's possible to first have a limited workshop with management only, to decide on drivers and identify abilities, and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



Explain current abilities with domain capabilities

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

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



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

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

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

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

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

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

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

Find a solution

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

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

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

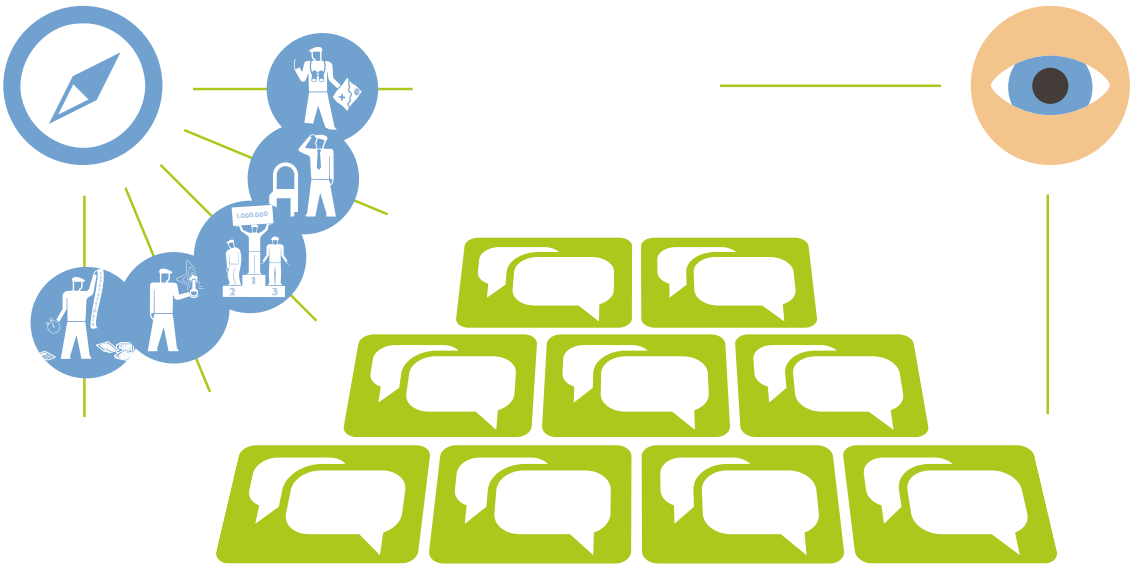
Make sure to evaluate different transition possibilities. Also make sure that all desired abilities have been addressed before the final set of transitions is defined.

Use the body of knowledge and experiences to be creative

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

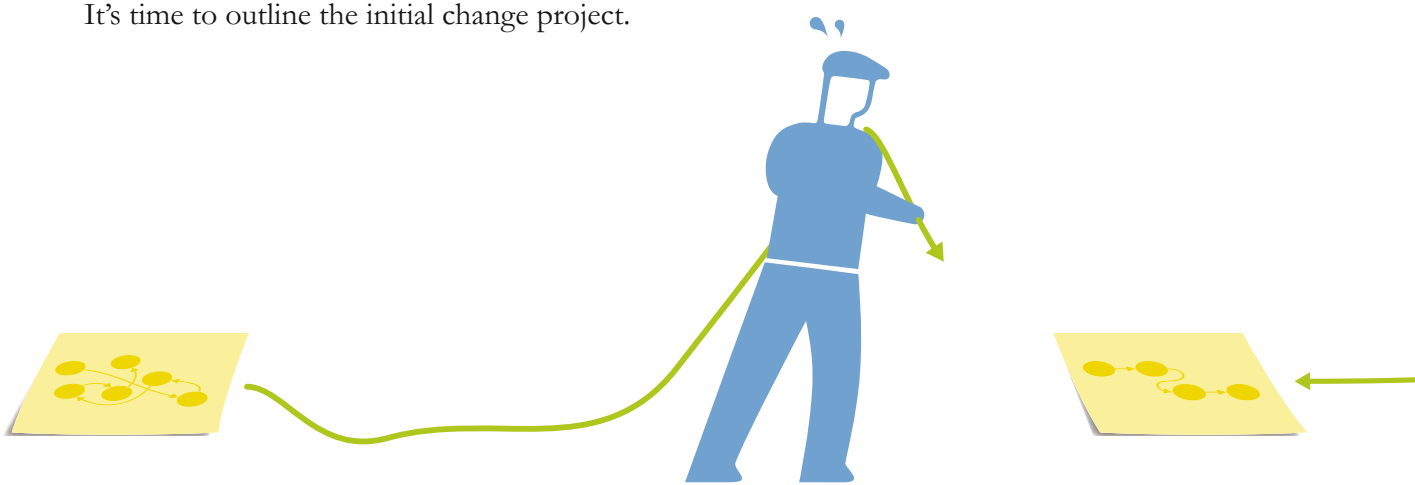
The best way to find relevant scenarios is to look at your drivers. In which one of the five main driver groups do you see your drivers. Then, go to the "The compass" part of the book to find relevant scenarios to read for your driver group.

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



Decision time

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



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

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

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

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


Done - let's get on
with the real work

The real work

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

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

Gartner Inc.

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

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

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

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

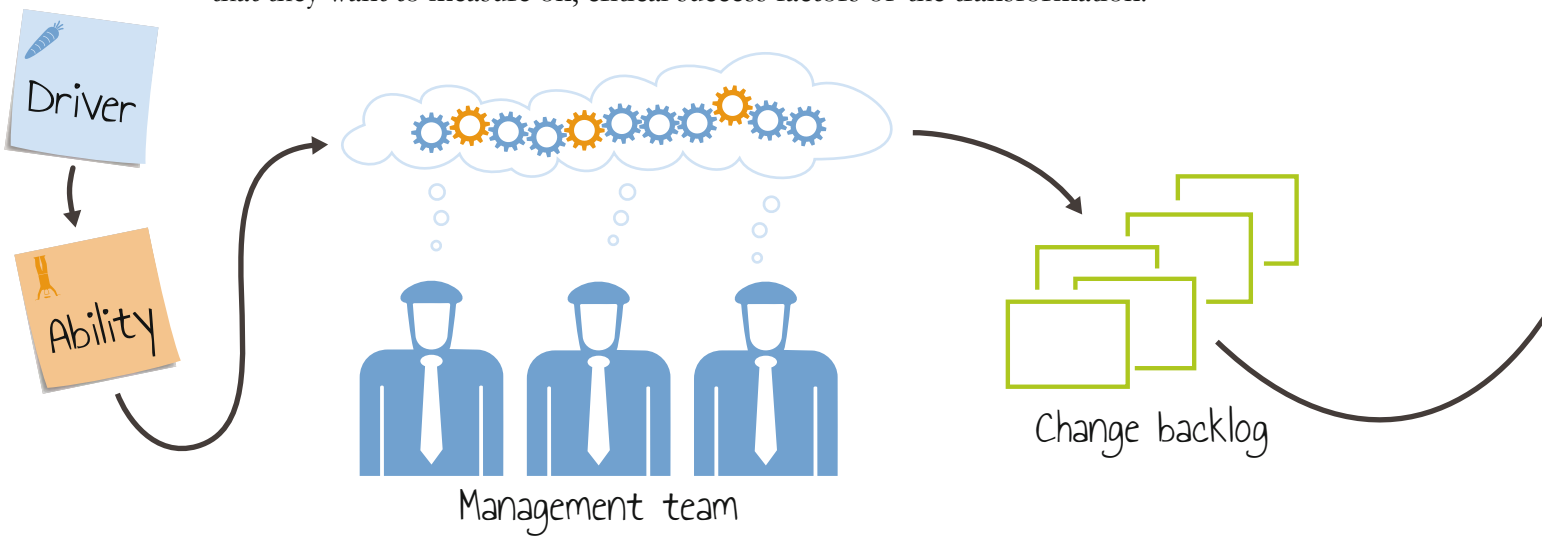
Agile change center

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

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

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

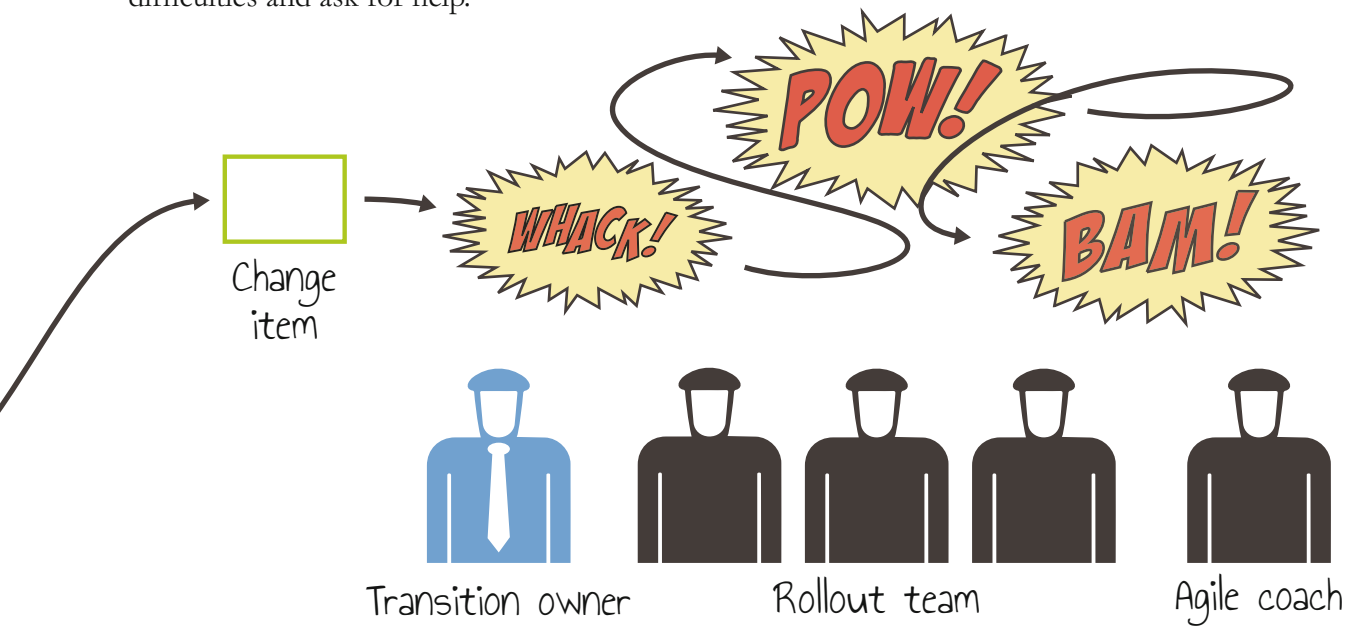
The center consists basically of two teams, the management and the doers. The management team defines drivers and prioritizes and aligns the overall transformation. They set all drivers, such as goals based on sales growth or return of investment. They have also identified abilities that they want to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



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

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

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



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

Alphabetical list of terms and definitions

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

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

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

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

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