Huge LeSS Huge at BMW Group — Autonomous Driving

Owners:  Konstantin Ribel and  Michael Mai

Companies Involved: valtech_ Valtech GmbH and Odd-e Odd-e

People Involved:  Markus Tecza,  Viktor Grgić,  Erwin Yükselgil,  Greg Hutchings, 
Frederic Siepmann,  Elisabeth Liberda,  Mark Bregenzer,  Nils Bernert, and  Daniel Heinen

Acknowledgements

We want to acknowledge the BMW Group for the opportunity to learn, grow, and analyze a fascinating LeSS Huge case in a highly competitive and challenging environment. We also thank all LeSS Trainers and Coaches who were present at the BMW Group during this journey.

We sincerely thank our mentors Mark Bregenzer and Viktor Grgic, who accompanied our development from the early days. And special thanks to Viktor Grgic for re-viewing and re-writing this experience report and always finding time to talk with us.

Craig Larman, thank you for your endless patience with us when (again) re-writing the case study and coaching us. Bas Vodde, thank you for your support, for continuously answering our (often nagging) questions, and supporting us on our way.

A big thank you to our colleagues and friends for their support!
And most strongly, we thank our families for their endless mental support on this journey.

It was and continues to be a deep and transformational learning experience for us.

Introduction

The BMW Group decided to go through a significant change to deliver the highest customer value, learn faster than competitors, and create an easily adaptable organization to ensure its first two goals. It is still in the middle of its journey.

This report is an in-depth examination of BMW Group's LeSS adoption in the autonomous driving department. It covers mid-2016 until October 2019, and the LeSS adoption is still ongoing.

The authors are Konstantin Ribel and Michael Mai. The real names of the people are left out for legal reasons.

The report is structured with multiple views (perspectives) on to the change:

1. Timeline View
2. Technology View

Start with any view you like!

Terms: LeSS and Scrum terms are capitalized, such as: Sprint, Product Backlog, Team. Note: *Team* is the role in LeSS whereas *team* is the general concept of a team.

Background

On BMW Group's journey as an automobile designer and manufacturer, its main focus is to provide more safety, comfort, flexibility, and quality through Level 3 autonomous driving (AD) (see Figure 1). There are many challenges to overcome on the way to AD, and perhaps the most significant is its inherent complexity.

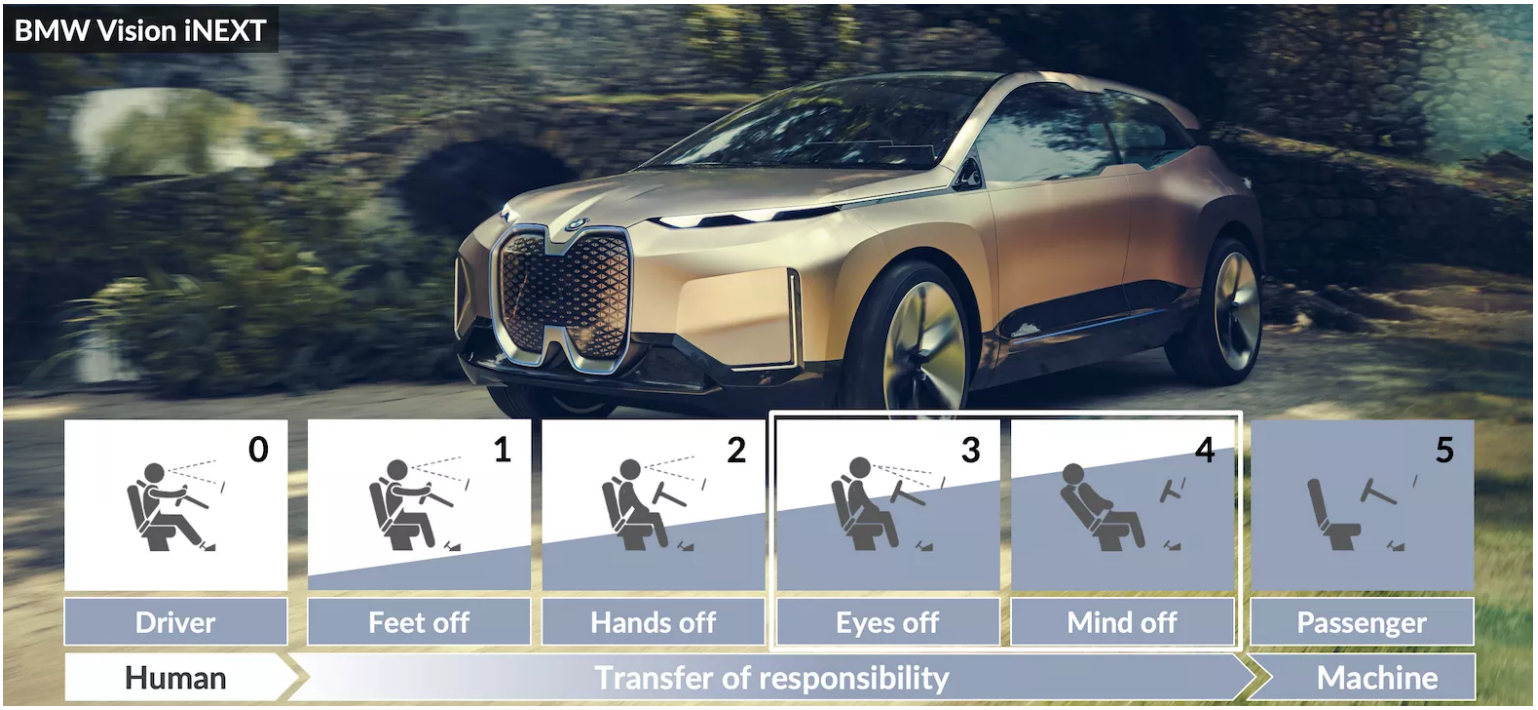


Figure 1: Autonomous Driving Levels.

To create something as complex and software intensive as AD, BMW Group has to shift from a 100-year-old company of mechanical engineering and manufacturing skill to a software company and treat AD as a complex software research and development (R&D) initiative. It sounds like a revolution and a paradigm shift. It is!

As an early step, BMW Group gathered all the people to be involved in AD and co-located them at *one* site—the BMW Autonomous Driving Campus. Why? Because they anticipated that succeeding in such a hard problem—combined with a paradigm shift—would be even more difficult if *multiple* sites were involved.

Then the BMW Group did what no one expected: they deeply changed their traditional organizational design towards one optimized for making learning and adaptation easier in a large group. And for that they took inspiration from LeSS.

So far, the journey has been full of failures and pain, but also full of positive effects such as closer collaboration between departments, leading to less inventory and handoff between groups of people. A higher focus on self-managing teams and frequent retrospectives led to teams designing and owning their processes. Higher emphasis on technical excellence led to better software design, cleaner code, and a more adaptive technical stack.

Timeline View

Three pronouns *I*, *we* and *they* provide different perspectives in the timeline view. It starts with the author's personal experience—*I* and *we* form. Then it transitions to the *they* form, presenting an external view on many situations, providing a more impartial observation followed by a root-causes analysis and suggested measures.

How it all began

After a successful launch of a new car in 2015, I (Konstantin Ribel) was looking for a new challenge. The Highly Autonomous Driving project had just commenced at that time. I joined the project at the beginning of 2016.

One of my main tasks was to create an efficient working model for developing our complex AD open platform, which involved BMW Group, cooperation partners, and vendors. For a standard complexity Advanced Driver-Assistance Systems (ADAS) Electronic Control Unit (ECU), we usually have a team interfacing with *one* vendor to develop it. For the AD open platform and its multi-processor hardware design, we estimated 5-10 times greater complexity, *several* vendors, and cooperation partners. We further discovered that if we scaled this up linearly, we would probably need 10-15 times more people solely for the vendors' project management interface (see Figure 2). I realized that something had to change to improve it.

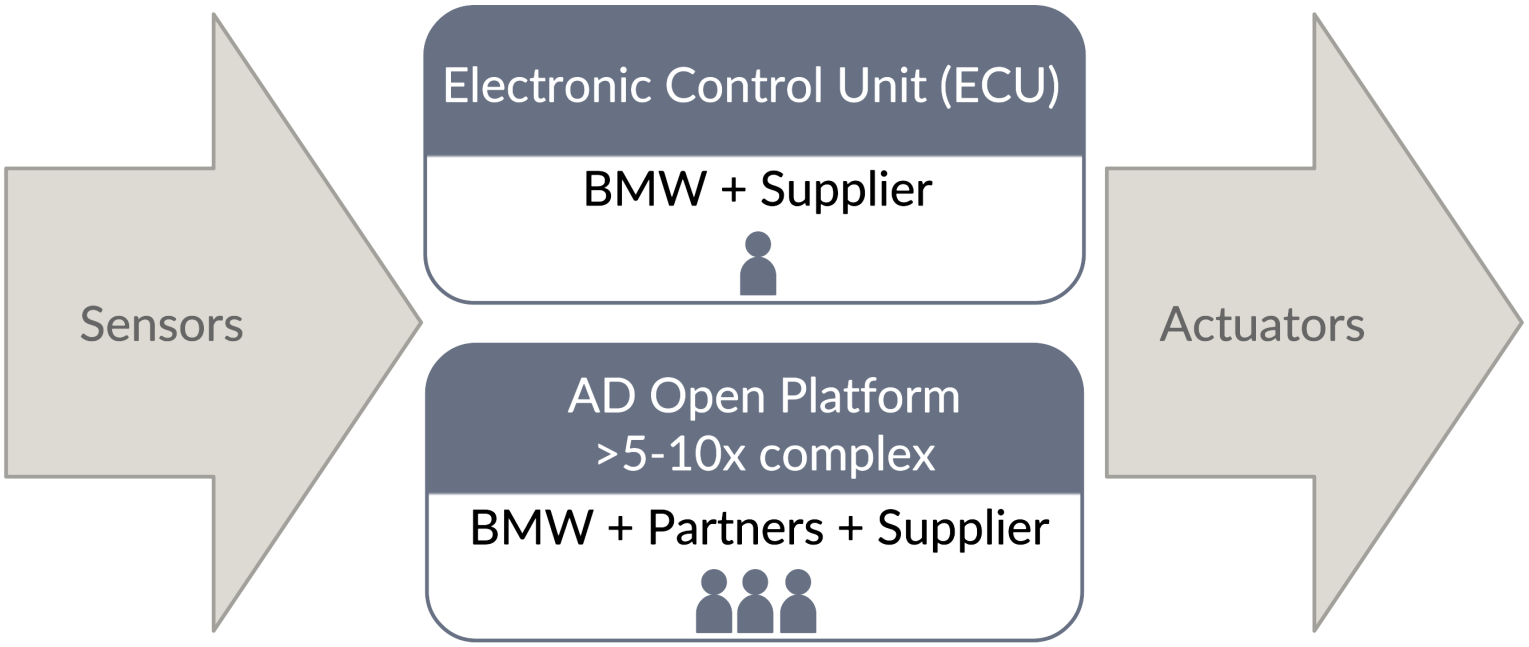


Figure 2: ECU schematic.

While looking for a solution, I read several Scrum and agile development books. Then I was recommended to talk to Mark Bregenzer—an agile coach and LeSS trainer. I met Mark and explained our situation. His answer was, "You need way fewer people than you can imagine. You need to change the organizational structure." After this meeting, Mark recommended that I participate in a Certified LeSS Practitioner (CLP) course with Craig Larman.

I participated in the CLP course in June 2016 in Berlin. On the second day of the CLP course, I realized that all of my roles at work manifested the lean wastes—"moments or actions that do not add value but consume resources." [1, p. 58].

Who wants their work to be a waste? I was devastated. After the course, I realized that I wanted to reduce the lean waste in the environment I worked in, and I was sure that my colleagues would want that too.

When I returned to the office, I tried to convince my line manager's superior ("Paul") that we should adopt LeSS in our department. Initially, this was anything but successful. We had a dispute about how this was never going to work in our environment. It was a very disappointing experience for me. However, I recalled Craig frequently repeating the following phrase during the CLP course: "Think and act like a politician [when introducing change], not like an engineer." I realized that I needed to act differently and stopped pushing this topic.

Let's take a look at this story from a different perspective.

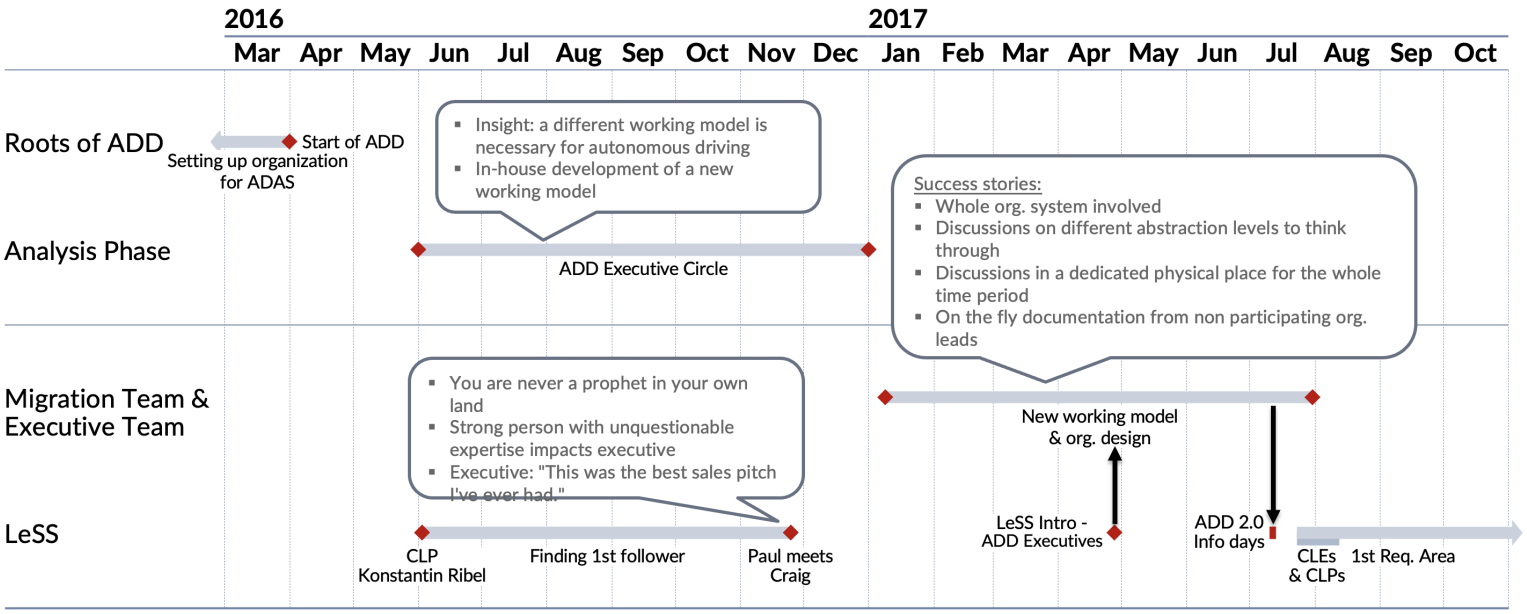


Figure 3: The bigger picture.

Between September 2015 and March 2016, an effort was made to set up an organizational unit for future ADAS systems, including AD. Executives of several departments worked together to accomplish this task. On April 1st, 2016, Several departments from different organizational units merged to form the new department for autonomous driving and ADAS (ADD—autonomous driving department).

A short time after ADD's start, there was an insight that within the existing setup, there was, among other issues, a high amount of coordination overhead and hand-off between departments, groups, and roles that were slowing us down. It was a weak starting position for a complex product with many uncertainties and unknowns, such as AD. To improve this situation, the ADD executive board worked between June and December 2016 on a new working model and organizational setup.

During this time, Craig Larman was on-site and taught our developers legacy-code TDD. I invited Paul to meet Craig at the end of November 2016. He took the offer. Craig and Paul spoke for about 20 minutes. Those who have met Craig know how honest and precise he can be. During their conversation, I did not know whether it was a good idea to let them talk to each other. Among other topics, Craig introduced Paul to Larman's Laws of Organizational Behavior and shared with him the English proverb "you are never a prophet in your own land." The further the conversation went, the more uncertain I became about its result. When Paul and I left the room, Paul said: "This was the best sales pitch I've ever experienced." It was a success! Paul became the first like-minded person on the executive level. The momentum for the change began to rise.

A so-called migration team, which I was part of, was founded in January 2017. To represent the whole organizational

system, this team consisted of managers and employees from different levels. It allowed us to discuss ideas through different abstraction levels. We continued to work on the working model that the ADD executive group had come up with the year before. We discussed different use cases, everyday situations, and how they would look if we adopted this working model. We were continuously working on the proposal to improve the situation we had.

Meanwhile, Paul convinced the ADD executives to have a 4-day workshop with Craig Larman on systems thinking and organizational design for large-scale agile development, also known as the Certified LeSS Executive course. To ensure full executive representation and due to their availability, this event took place a few months later. To get the ball rolling before then, in April 2017 I organized a one-day introduction workshop with Mark Bregenzer.

Figure 4: Mark Bregenzer is debriefing a systems thinking exercise.

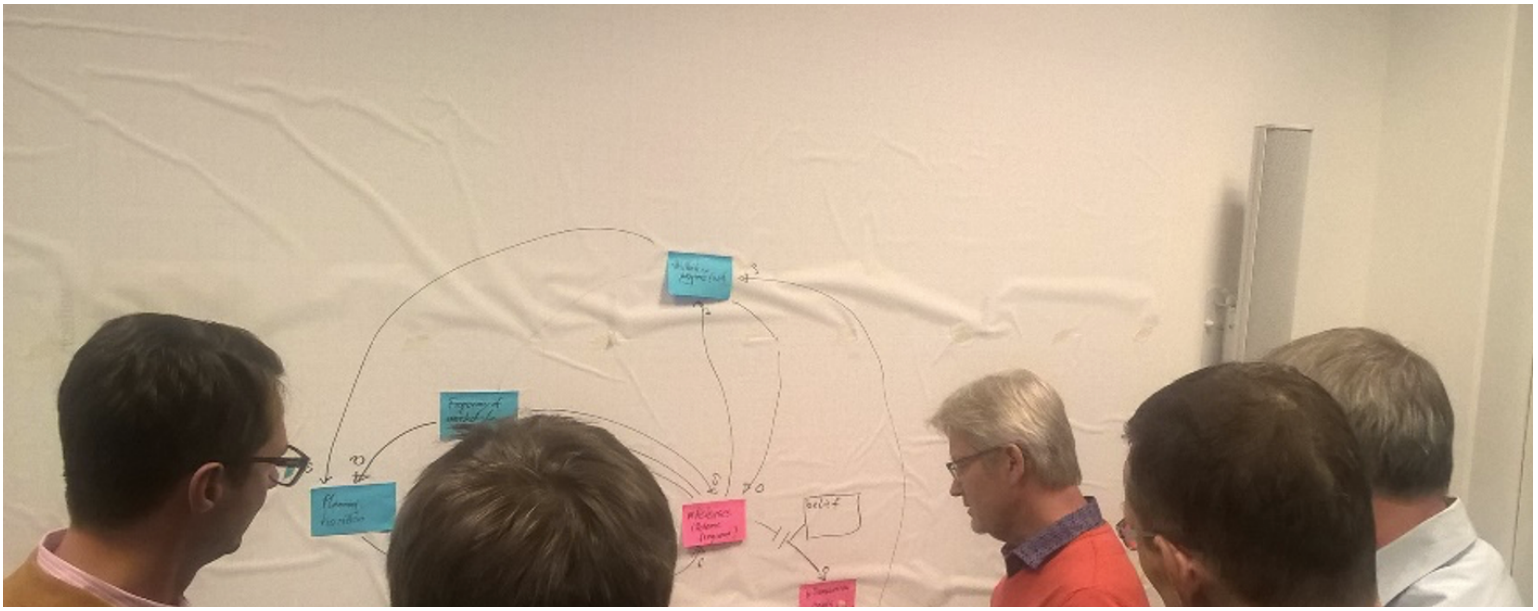


Figure 5: Executives are learning and exercising systems thinking.

The content, presented by Mark, inspired participants. It was evident that we needed to move in this direction. This initial workshop created the next significant momentum shift.

At this point, the ADD executives were certain that to really change the behavior and the organization's working model, they needed to change the organizational *structure*.

Summary of Beginning Lessons

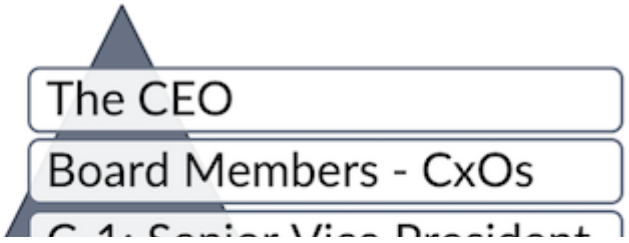
- Think and act like a politician, not like an engineer
- Gain allies across different hierarchy levels and align your shared goal—real change is only possible as a group, both top-down and bottom-up. LeSS guide: Three Adoption Principles [3, p. 55-59].
- Educate all senior executives and directors, especially those with true decision-making powers [3, p. 57].
- "You're never a prophet in your own land." Engage an external expert, experienced in large-scale organizational design for agile development. This expert—a great trainer and a great coach—will have a focus on the *Why* and will make a positive difference in your LeSS adoption. More on this topic in the section "Teach why" in the book Large-Scale Scrum [3, p. 60].
    - (LeSS guide: Getting Started)

Preparation Phase

The Way to the Insight that Organizational Change is Necessary

> LeSS adoption involves big organizations and many minds with deeply rooted assumptions about how organizations should work. Successful adoption requires challenging these assumptions and simplifying the organizational structure, with all the explosive politics and "loss of face" that working across a big group entails. Adoption needs everyone to improve towards a shared goal. [3, p. 54]

Our starting position was a common hierarchy (see Figure 6).



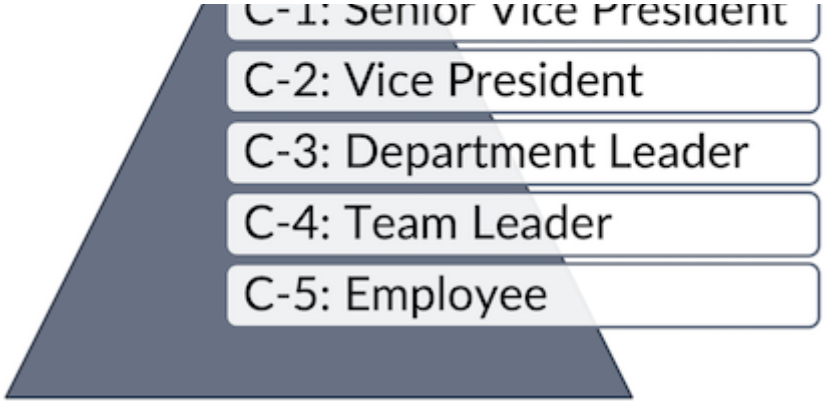


Figure 6: Typical hierarchy.

ADD inherited from its origin departments more than 15 different roles with defined interfaces, clearly describing where someone's work starts and ends. With this setup, we successfully delivered many great cars to our customers. However, it is fertile ground for the lean-thinking wastes, such as hand-off, coordination overhead, and many others. Since waste inhibits adaptivity in complex product development, especially large-scale, we had a compelling motivation to change.

This phase started in May 2017, directly after the one-day introduction workshop with Mark Bregenzer. At this point, the answers to the following questions were open:

1. How do we want to work?
2. Which working model are we going to adopt?
3. If it will be Scrum-based, which scaling framework is it going to be?

To answer these questions, we needed to include the whole organizational system, including all relevant stakeholders from the rest of the BMW Group. We set up two teams. The existing migration team was too large, and therefore we shrunk it. The second was the executive team with a maximum of 10 members, as shown in Figure 7.

With this setup, we involved people who were highly engaged in product development and managers with far-reaching decision making powers. It allowed us to verify ideas and make decisions quickly.

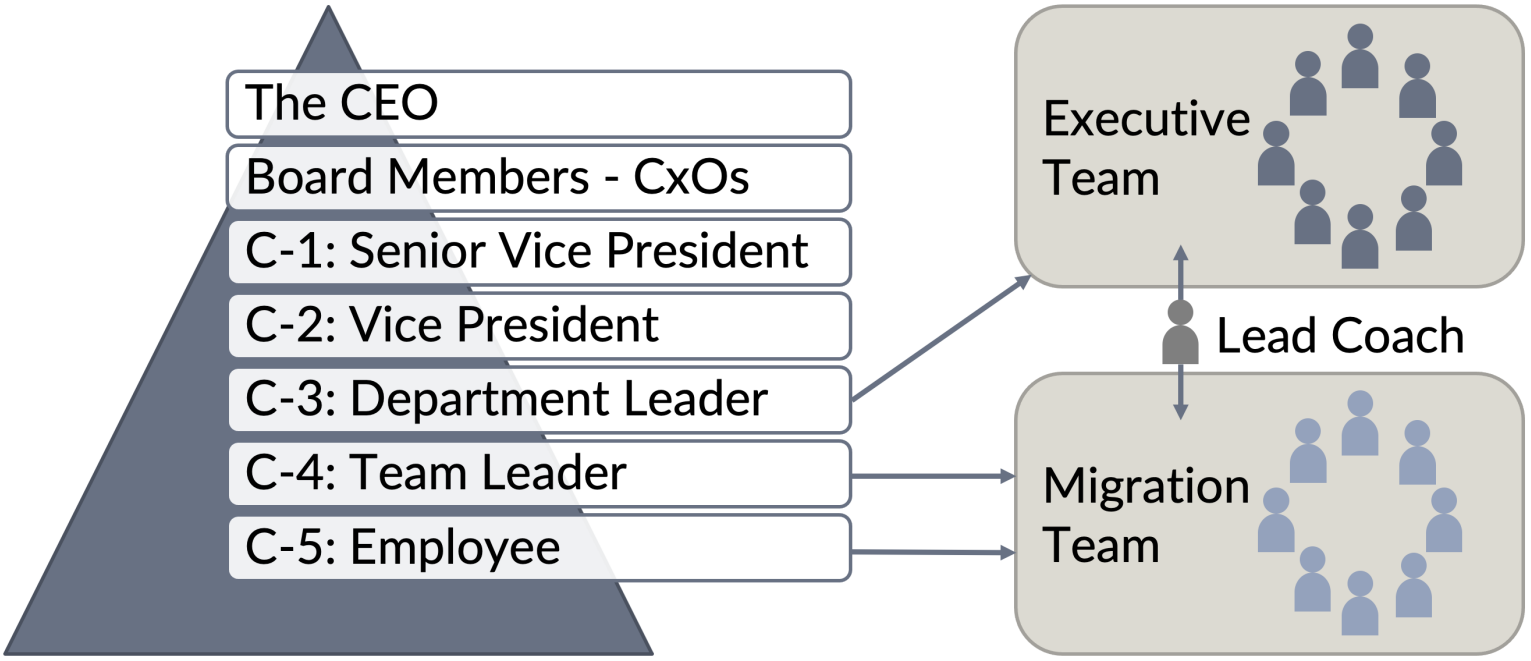


Figure 7: Teams covering the whole organizational system. Stakeholders and interface partners not shown.

Further, there was Mark Bregenzer, who challenged our assumptions about organizational structures and coached us on the Why. We worked full-time on this topic.

Both teams and Mark worked in *diverge-merge* cycles. The migration team and executive team representatives explored answers to the question: How do we want to work? In a multi-team setup, both teams debated the rationale behind possible solutions. The representatives worked out organizational designs which hypothetically would support the proposed solutions. Then, the whole ADD board (C-1 and C-2 level) and the executive team reviewed both teams' proposals, meaning the answers to the "How do we want to work?" question and the supporting organizational design.

Both teams worked closely together, allowing fast feedback and short iteration times for their ideas.

Figure 8: Close cross-team collaboration across functions and hierarchies.

A few weeks later, after both teams gained many insights, the organizational design's optimization goal became apparent. First, no one in the world knew *exactly how* to make an autonomous-driving car production-ready. Therefore, learning *what* and *how to do it* becomes vital. Consequently, it requires an organization to become a *learning organization*.

Second, faster feedback loops improve learning. Since no one had yet built an (SAE Level 3 and higher) autonomously driving car, learning from reality, adapting the Product Backlog, and working on the highest-priority items was necessary for any hope of succeeding. That meant short cycle time, early integration, and prototype cars driving on public roads. In other words, using feedback loops to learn and then to decide what to work on next. Consequently, the ability to always work on the newly discovered most critical items requires an adaptive organization.

The above led to the following optimization goals:

1. Ability to learn faster than competitors
2. Based on that learning, ability to work on and deliver the highest customer value
3. Easily adaptable organization to ensure (1) and (2)

It was necessary to remove waste to achieve the optimization goals. First, coordination overhead needed reduction. For example, identifying the right people with experience and expertise for focused discussions, and particularly excluding middle management, to avoid individual tactical career games that disturb these discussions and usually increase the overhead.

Second, hand-off waste, especially between component teams, single-function silos, and individual responsibilities, needed removal. The guiding goal was: increase shared responsibility across functional groups.

We thoroughly analyzed whether all of it was achievable with LeSS in our domain. For this analysis, we invited people who were not part of the migration and executive teams, who had different roles and use cases. We let those people challenge the LeSS working model and thoroughly discussed their use cases, such as defects handling and coming to a driving approval for public roads in the context of safety-critical software and shared common code. The discussions focused on how we can achieve the use case results with LeSS.

We were looking for use cases where it was not possible. We didn't find any.

The executive team had in addition to LeSS other proposals on how to achieve the result. Only *one* required any changes in the organizational *structure*—the hard way. The others just needed changes in *practices*—the easy way. The executives already learned that just changing the practices without restructuring the organization design does not change much of anything. Some executives called those attempts "more of the same" with different labels.

In contrast, LeSS required deep changes in the organizational structure—the harder way, but more likely to support the goals. The decision was to adopt LeSS (June 2017).

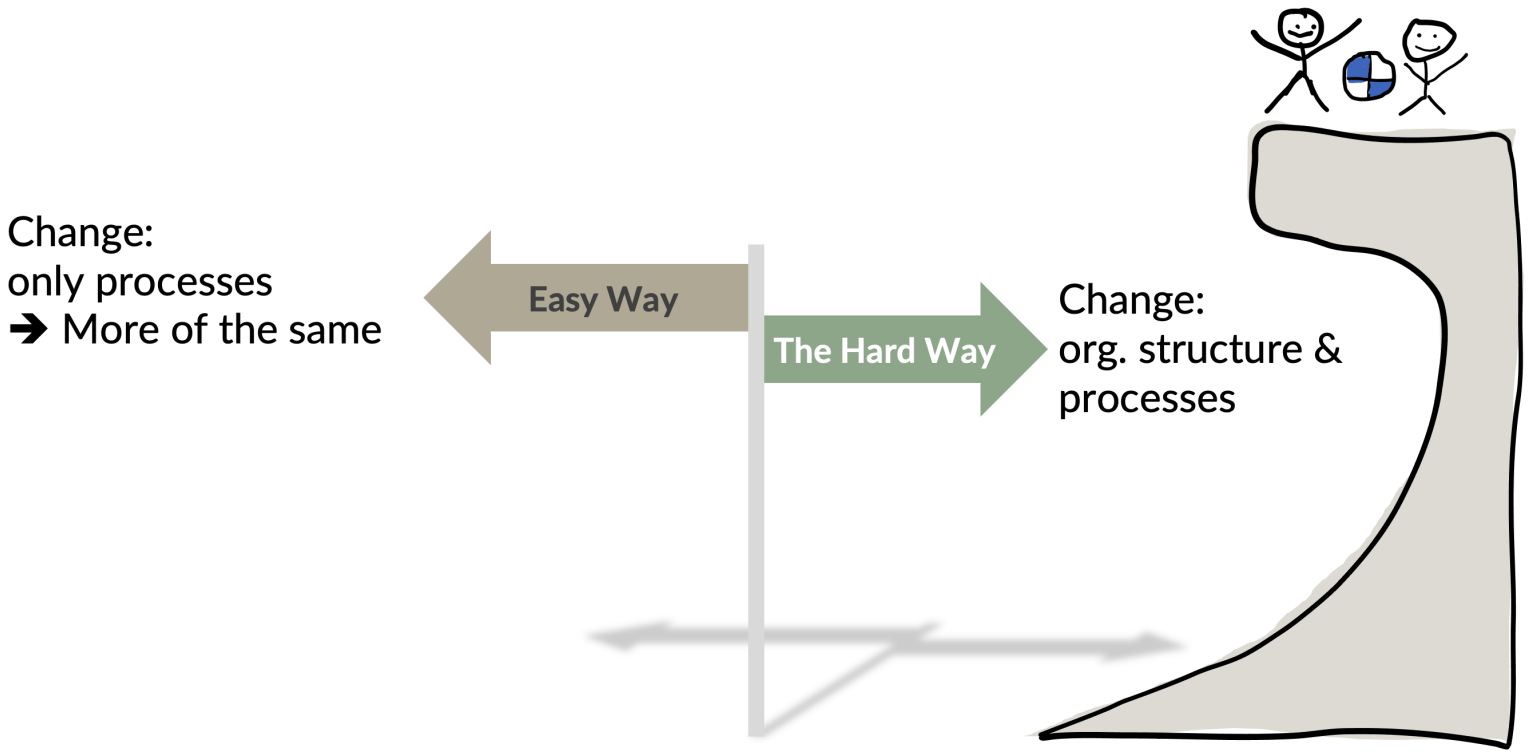


Figure 9: Easy vs. hard way. On the right: a climbing wall with an overhang.

Product Definition

AD is driven by customer demands and legal requirements. New technologies and seamless connectivity are paving the way. The technological challenges are enormous. Building up data centers, training AI algorithms, virtual test and validation, developing new sensors from scratch, bringing non-automotive high-performance hardware to the car and qualifying it for automotive use, and complex system architecture just to name a few.

Figure 10: Technological challenges.

What is the Product for This LeSS Adoption?

> The product definition determines the scope of your Product Backlog and who makes a good Product Owner. When adopting LeSS, it determines the amount of organizational change you can expect and who needs to be involved. [3, p. 157]

A customer would most probably describe AD as a mobility service spanning a range from a smartphone app, over mobile networks, cars, sensors, algorithms, actuators, tires to road infrastructure. The list is long, and this is just a technical perspective. From a legal perspective, there are insurance questions to deal with, road clearances, law changes, etc. Many parties need to be involved, to live up to this product definition. In addition to BMW Group, there are governments in each country, insurance companies, vendors, service providers, and many others. The customer perspective might be correct, and simultaneously, it is unfeasible to start a LeSS adoption at this scale.

There are forces that restrained the product definition:

1. Organizational boundaries for the LeSS adoption;

2. The parties which can be involved

Both forces naturally restrained the product definition to a car with an autonomous-driving feature where the BMW Group was the main party. This product definition required collaboration with other departments within the BMW Group and vendors who did not adopt LeSS.

Figure 11 provides an overview of the organizational structures within the BMW Group. Each department within R&D had its dedicated senior VP. The ADD senior VP sponsored this LeSS adoption, which naturally limited it to the organizational boundaries of ADD. Consequently, existing interfaces and working models to other departments and parties outside the ADD needed to remain intact.

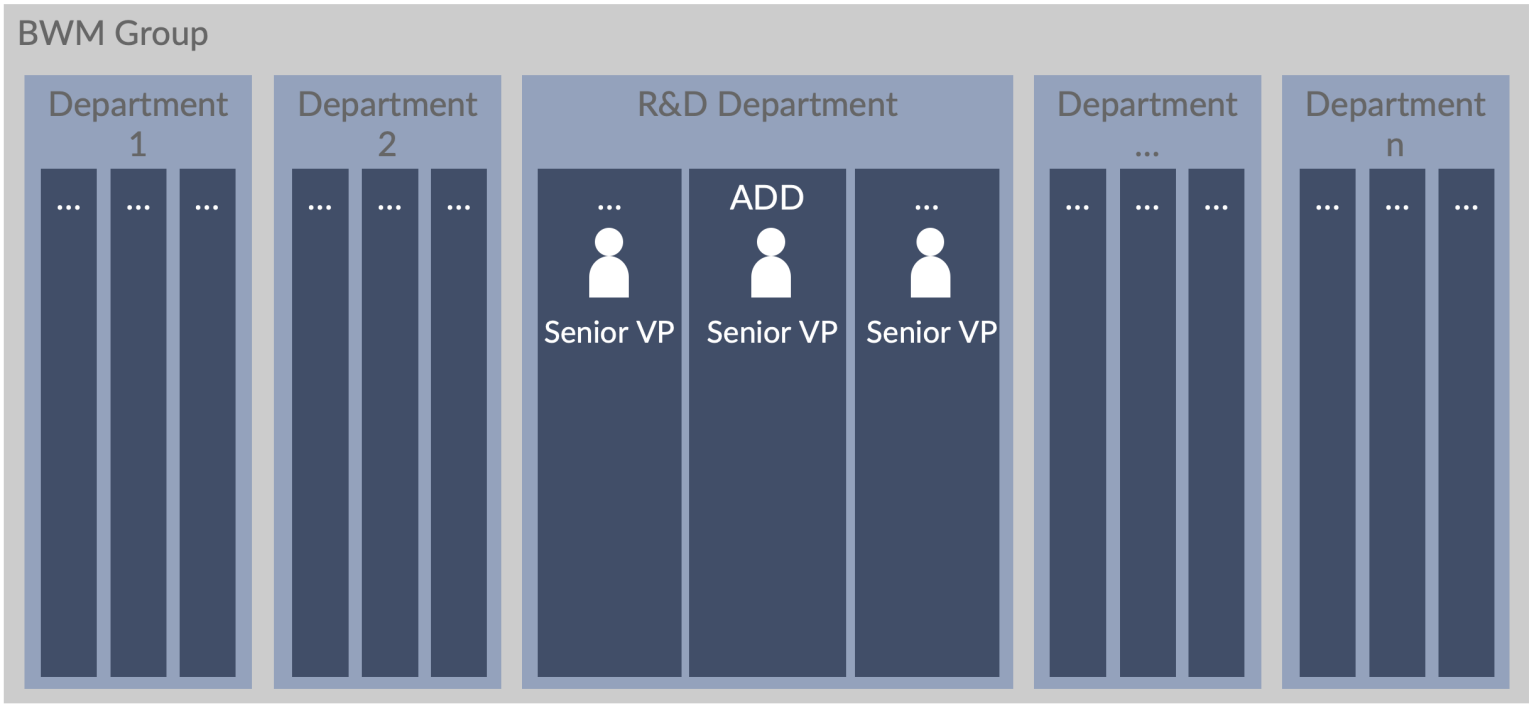


Figure 11: Boundaries for the change.

> Most product development is organized as projects—every new product release is a new project. Organizations manage development by managing projects ... [1, p. 236]

This quote describes perfectly the situation at BMW Group. Let's take ACC (Active Cruise Control) as an example. The BMW Group released its first version of ACC in 2000. With each further release, ACC gained new improvements and could handle more and more traffic situations. The product enhancements over multiple releases make it a product development case.

A term needs definition before continuation.

Start of Production (SoP): A major milestone. It involves an updated or new car model whose production will start after this release. Further, the technology stack (hardware and software) is updated.

Despite the long-lived products, the SoPs were organized as projects.

ADD had two big projects, SoP 2018 and SoP 2021. To ensure stable delivery for SoP 2018, involved people were excluded from the LeSS adoption. They could join the LeSS organization after the release. Therefore, their organizational setup remained mostly unchanged.

This decision also excluded the SoP 2018 product scope from the possible product definition.

As a consequence, the product definition for the LeSS adoption became the release scope of SoP 2021, excluding the SoP 2018. Mainly ADAS functionality on SAE Level 2 and AD for SAE Level 3. It involved the development of sensors, computing hardware, and software. It also required collaborating with departments and vendors outside ADD, who did not adopt LeSS.

Further, AD should be offered as a separate Autonomous Driving Platform (ADP) to other automobile OEMs.

Establish the Complete LeSS Structure At The Start

LeSS requires small, end-to-end, self-managed teams coordinating towards a common goal while sharing the responsibility to achieve it. Our starting position was far away from the intended one (see chapter The Way to the Insight that Org. Change is necessary). Further, our mindset, which formed over decades, supported single-function groups, heroism, component teams, and only one career path that required a switch from technical to a management career. We had concerns that without changing the rewards system, people would not change. Therefore, we faced the question, "If, when people cooperate they are individually worse off, why would they cooperate?" [6].

It led us to the insight that we needed to create an organizational design which would foster a culture "... in which it becomes individually useful for people to cooperate" [6]. You can find more on this topic in the Culture Follows Structure guide.

The LeSS Rule "... establish the complete LeSS structure 'at the start'" was followed. We designed our new organizational structure guided by the typical LeSS Huge organizational chart (see Figure 12).

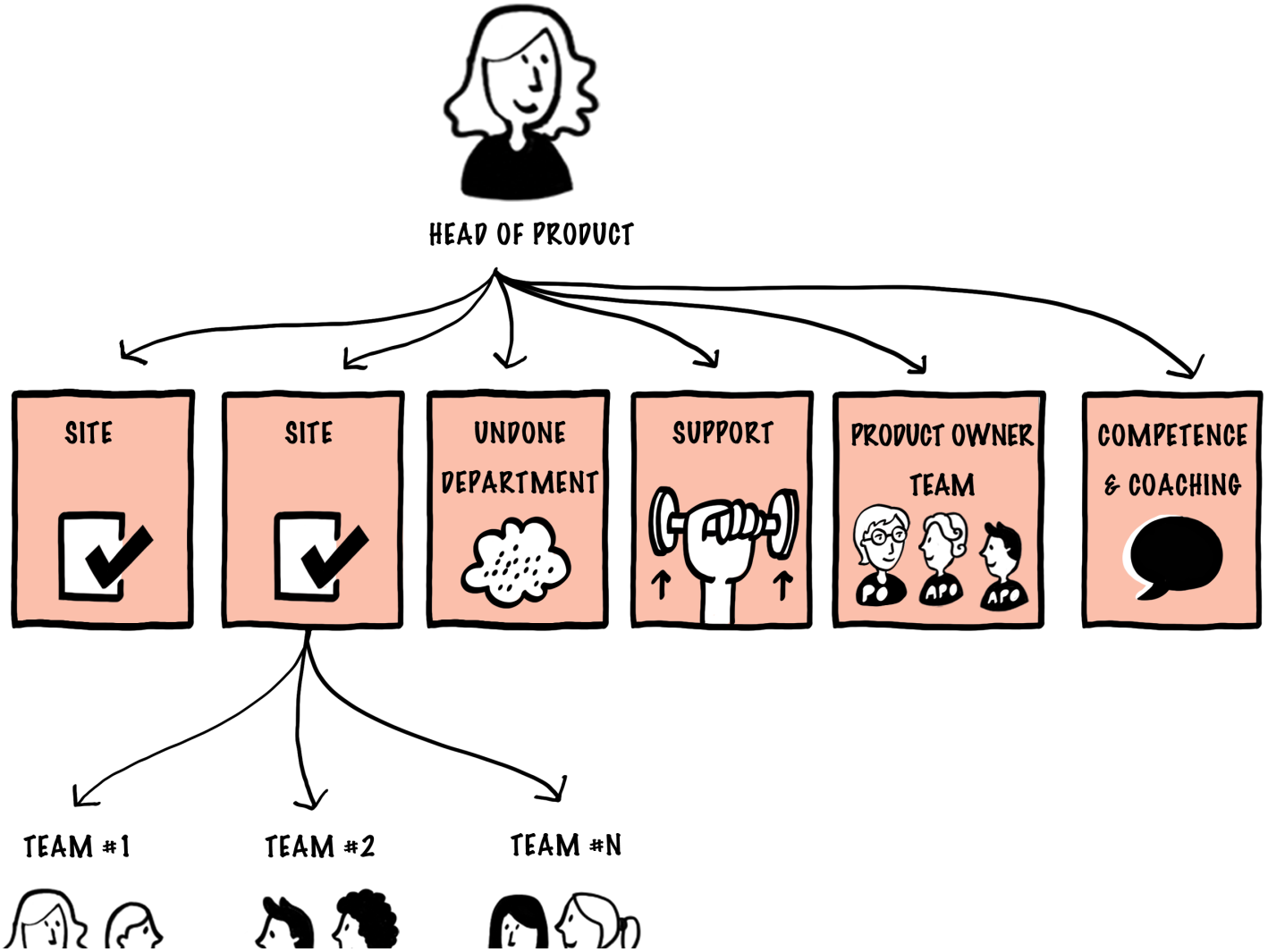


Figure 12: Organizational structure for LeSS Huge.

Both the migration and executive teams played an important role in inspecting previous organizational structure, and applying systems thinking to change the organization towards the “learning and adaptiveness” goal.

We had concerns that any cross-functional and team-based organization would, over time, inevitably revert to the old paradigm of single-function groups. The deeply ingrained BMW culture and structure, which was very different from the desired one, was (and remains) a strong force that reaffirmed our concerns. We decided to resolve this by making significant and immediate changes in the structure (see Figure 13). We adopted this to counter the tendency to fall back into old habits. The following paragraphs explain each part of the structure.

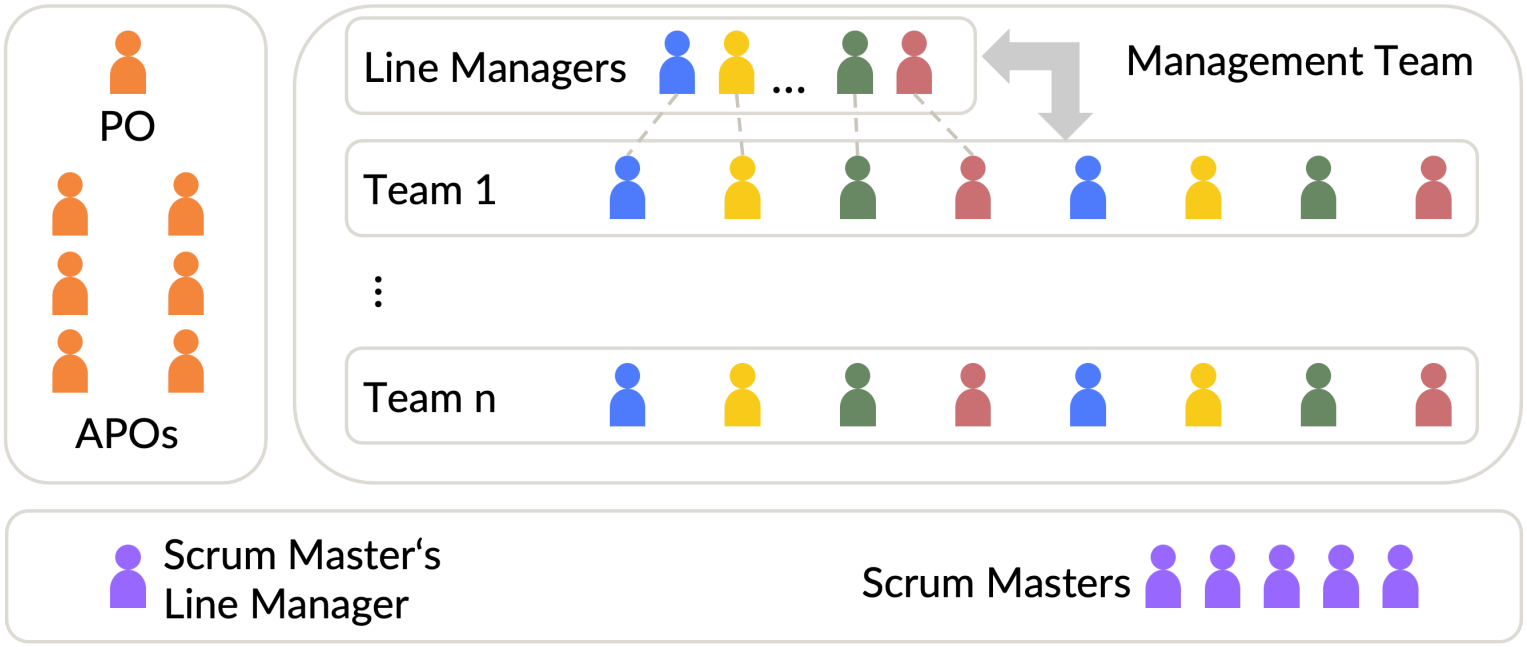
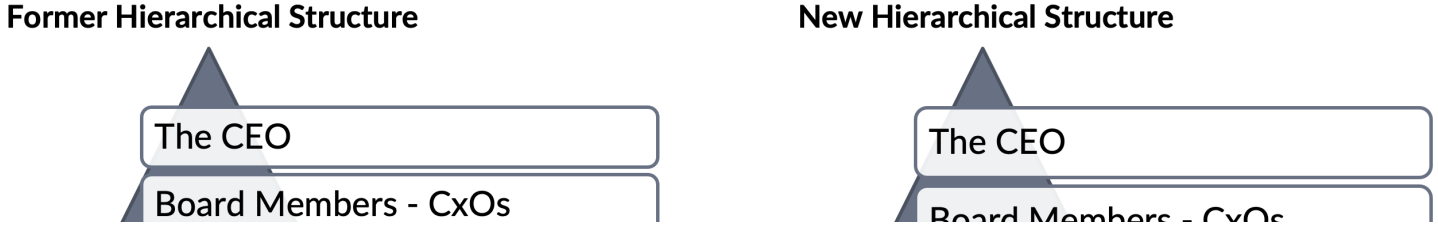


Figure 13: New organizational setup.

Step 1 - Set up a Development Department (Feature Teams and a Management Team)

This step starts by removing one full hierarchy level, the role of team leaders (C-4 level, see Figure 14), which leads to a remaining group of line managers each being disciplinarily responsible for approximately 60 product developers. Which, in turn, creates several opportunities to form cross-functional and cross-component teams.



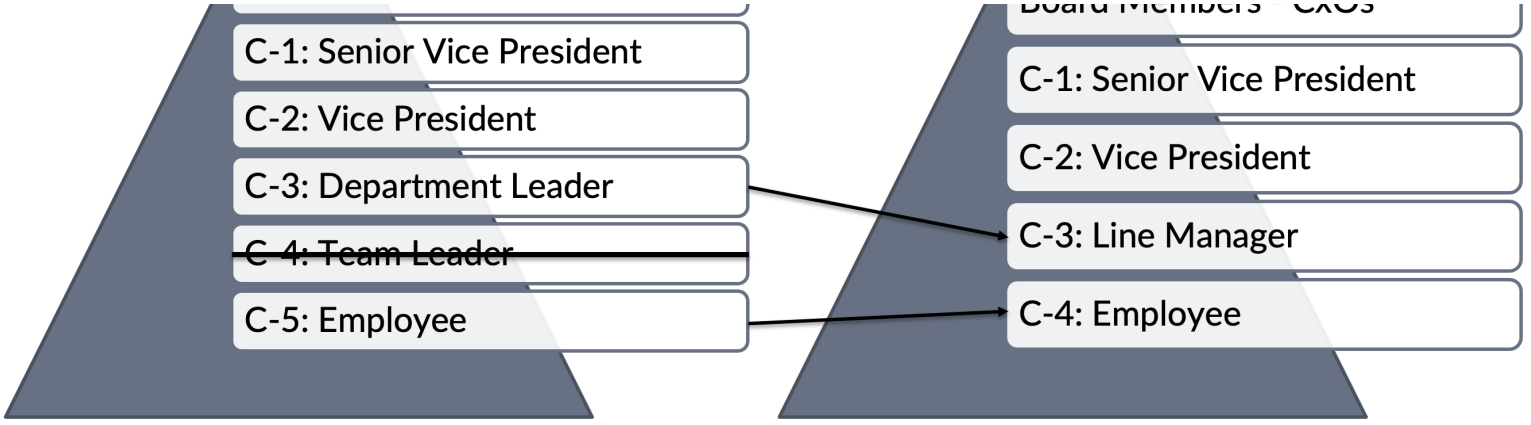


Figure 14: Role of team leaders removed.

How to set up connections of feature team members with line managers (which was required at the BMW Group)? We considered two options.

Option 1: Feature Team Members Having the Same Line Manager

One option was to arrange the eight teams (one Requirement Area) per each line manager.

Advantages? We couldn't think of any except variations of local optimizations.

Disadvantages? The concept of Requirement Areas demands *flexibility*. They must be *easily* set up and dissolved. It must be *cheap* to reassign a team from one Requirement Area to another. However, this option brings a degree of organizational stiffness with it. If a Requirement Area is an organizational unit, reporting to one line manager, then it would require formal organizational changes to, for example, reassign a team from one Requirement Area to another, or dissolve the Requirement Area.

Other disadvantages when considering this option over time and scale:

When coming closer to a release date, delivery pressure would increase. Then, old behavioral patterns could revive. The managers could be made "responsible" for their teams' performance, leading to command-and-control behavior. In this situation, managers would be likely to focus on the teams rather than on the *system for the teams*. This, in turn, would lead to managers interfering with teams and their Scrum Master's area of concern.

Further, in the previous setup, the future line managers were single-function group managers (e.g., manager of the architecture group) on C-3 level, having team leaders on C-4 level reporting to them. Therefore, they were likely to be biased towards (1) their original single-function activity and (2) building up an informal layer of team leaders (even though the *official* position of team leaders was removed). Why would they do that, and why is this a problem?

First, note that promotions would remain manager-driven. And climbing up the career ladder at the BMW Group requires at some point to demonstrate management skills. Therefore, it's possible that employees attending more to the specialty of the manager (e.g. architecture) might gain more favor, and even be informally supported by the manager to take a more active "leading" role in their team. This could create an informal hierarchy within an ostensibly self-managing team and "…destroy the team's shared responsibility and cohesion." [3, p. 63]

Option 2: Feature Team Members Having Different Line Managers

The second option's main goal was a high degree of flexibility in changing Requirement Areas as needed.

We considered teams with developers having different line managers (see Figure 13). With this setup, each line manager would have around 60 product developers distributed over many teams.

Advantages? A high degree of flexibility to easily change Requirement Areas without changing formal organizational structures.

Further, this setup would expectedly force line managers to optimize globally. How? This option would *equally* authorize line managers towards the teams. It would likely reduce the effect of managers using their authority towards a specific "my team" to optimize locally. It would create the context that the line managers act as an aligned management team towards the teams—a positive application of the "culture follows structure" pattern.

Disadvantages? None obvious to us, but there were open questions. Previously, the org-chart reflected single-function groups and personalized responsibilities, which made it easy to find someone for problem X. With cross-functional teams and shared responsibility, this information was not visible in the org-chart anymore. How would other BMW Group departments, which remained in a traditional organizational setup, communicate with ADD?

Further, how to escalate and to whom, when the responsibilities become shared and the organizational structure detached from the product architecture?

Existing LeSS guides can address those questions—for example, *Leading Team* [3, p. 308] who commit to a long-term topic, for example, collaboration with an adjacent department. Since ADD did not experience those practices before, it was hard to imagine how they would work out.

We chose to leave those questions open and answer them using inspect & adapt.

The advantages were so important to us that we decided to start with this setup.

Step 2 - Set up a PO Team

Who should be the Product Owner? As described in the guide Find a Product Owner Given Your Type of Development [3, p. 173], the first step in finding a PO is to know your *development type*.

On the one hand, our product—AD—is part of a larger product, the car. In that sense, it was like internal component development. On the other hand, the paying customer needs to add the AD feature to the car configuration and pay extra.

The pivotal point was that the original vision, first set by the head of ADD, was to also offer AD as a separate Autonomous Driving Platform (ADP) to other automobile OEMs. In those ways, it was an external product itself.

For external products the LeSS guide Who should be Product Owner? [3, p. 173] recommends looking for a PO in the Product Management department, such as head of Product Management. Further, the guide defines:

> As Product Owner, you have the *independent authority* to make serious business decisions, to choose and change content, release dates, priorities, vision, etc. Of course, you collaborate with stakeholders, but a real Product Owner has the final decision-making authority. [3, p. 175, emphasis added]

Several factors constrained the PO choice.

The BMW Group had (and still has) a wide range of products, consisting of complex systems. The development and maintenance of which involve tens of thousands of people. The company's structure reflected the car's sub-systems, which increased its complexity and fragmentation over the years, especially with the increasing importance of software in each sub-system.

Therefore, no one has the required independent authority to make *or change* serious business decisions, not even the CEO. Instead, all serious business decisions are made jointly in committees. Therefore, we knew that the independent decision-making authority of our future PO would be somewhat limited.

There were two rationales for choosing a PO from within ADD: (1) the vision of offering the ADP to other automobile OEMs and (2) the fact that LeSS was being adopted at ADD, not at the whole BMW Group scale.

The head of a previously-existing department was already responsible for all ADAS and AD *customer offerings*; he also held the budget to develop them and decided how to spend it. In the sense of independent decision authority, he had, not the ultimate, but a lot of it. Of course, he needed to align with stakeholders and consider the entire BMW Group's product range when making decisions. This person became the PO. The APOs were recruited from different parts of ADD.

Unfortunately, as will be explored later, "a lot of" the authority was not enough to establish a properly-structured and prioritized Product Backlog.

The PO and APOs formed the PO team.

Step 3 - Set up a Competence and Coaching Department

> Software is created by people. Improving people improves products. [3, p. 111]

Relentless training and coaching are necessary to achieve mastery in any aspect to improve the product. To support this idea, we created a Competence and Coaching department, as described in the guide Organizational Structure for LeSS Huge [3, p. 111]. It consisted of Scrum Masters, technical coaches, and process-related staff, such as industry-standardization experts.

Why were the Scrum Masters in this department? The Scrum Masters' responsibility is to teach Scrum (and LeSS) and how to derive value with it, coach the teams, the (A)PO, the line managers, and the whole organization applying it, and last but not least, to act as a *mirror*. This sort of coaching, especially when acting as a mirror, requires everyone to be on the same conceptual level as the Scrum Master. It is even more critical when previous career paths and grading of individuals influence the perception of different roles and their hierarchy levels.

The typical case at the BMW Group is that the higher the salary grading, the higher the person is in the hierarchy. The grading usually comes with experience and demonstrated skills. Most Scrum Masters had lower grading than APOs and line managers. This raised two questions:

1. How will the Scrum Masters perceive and approach the higher graded APOs and line managers when coaching?
2. How will the APOs and line managers perceive the "less experienced" (based on their grading) Scrum Masters?

And let's also add the IPA (individual performance appraisal) to the coaching perspective. One's direct line manager conducts the yearly IPA. It results in changing one's grading and salary. Further, it is up to the line manager what data they use for the performance appraisal.

Consider the following hypothetical situation. Suppose a few teams and their Scrum Master have the same line manager. The Scrum Master observes that anti-patterns concerning self-management underpin the line manager's behavior. What will the Scrum Master do?

1. Will the Scrum Master act unbiased as a mirror towards the line manager and make this behavior transparent, risking hurt egos and risking an adverse IPA. *Or*? …
2. Will the Scrum Master accept this behavior mostly or entirely favoring a better IPA for herself as a Scrum

Master?

During the discussion of the new processes and the organizational structure, especially the Competence and Coaching department, we reasoned that most Scrum Masters would be biased and would choose the second option. But we wanted to create an environment where Scrum Masters could act as fearless and as unbiased as possible.

Further and considering the above, BMW Group employees were not accustomed to directly talking to higher management without consulting their direct manager beforehand. Therefore, the risk that top-management would get filtered information instead of the real picture from Scrum Masters was considered harmful.

To address these problems and create an environment where Scrum Masters can carry out their job as free as possible, we explicitly wanted the Scrum Masters to have a different line manager than team members. This decision led us to place the Scrum Masters in the Competence and Coaching department.

The Result

The resulting organizational chart looked as shown in Figure 15.

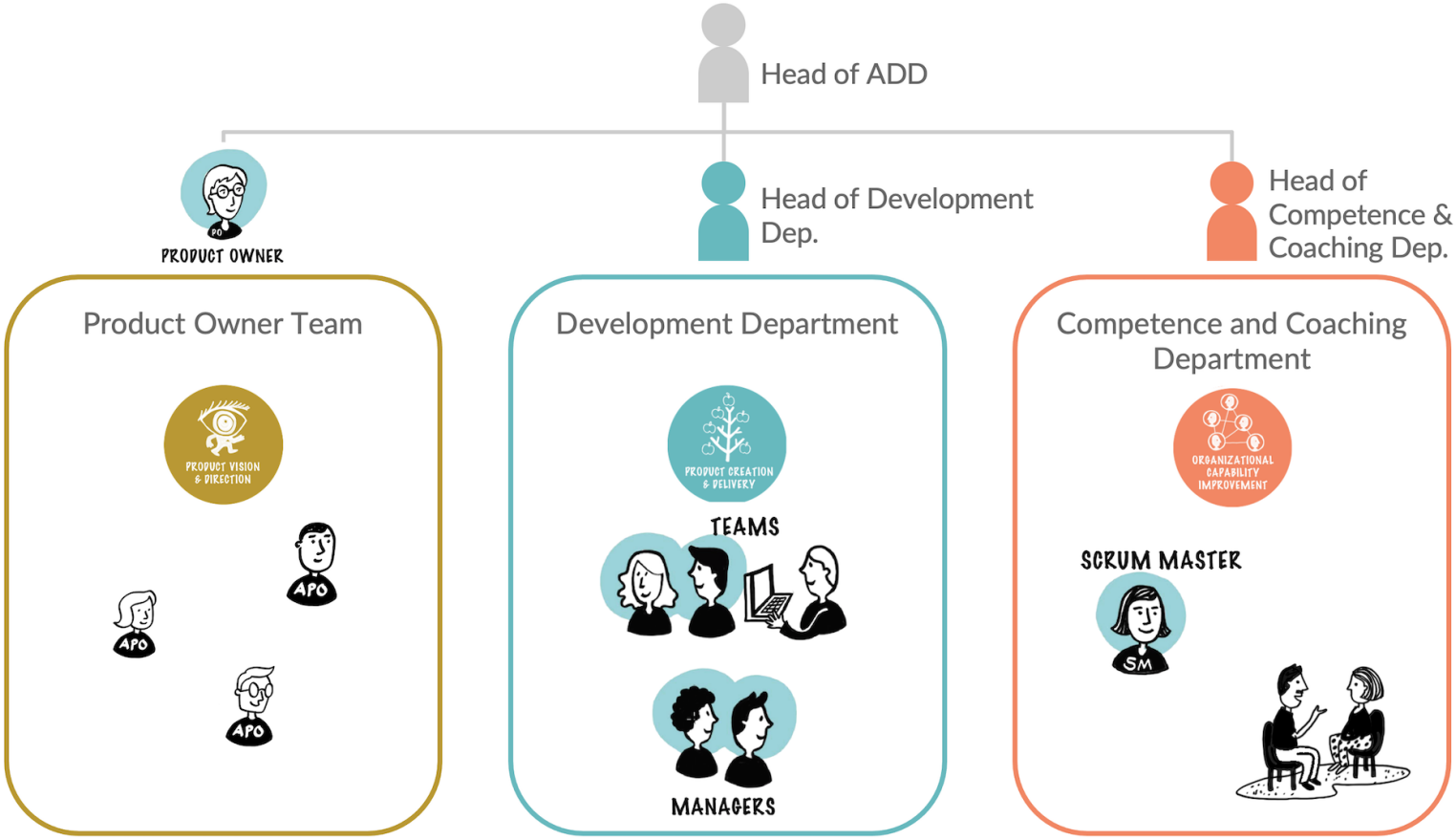


Figure 15: Resulting organizational structure.

There were three main departments within ADD. The PO, APOs, and their supporting staff comprised the Product Owner department. All Product Developers and line managers were in the Development department. And the Competence and Coaching department contained all Scrum Masters, technical coaches, and additional industry-standardization experts.

Those three departments together sourced the Requirement Areas. APO from the PO department, the teams from the Development department, and the Scrum Masters from the Competence and Coaching department.

A New Age—the First Requirement Area—Begins

The first of the Three Adoption Principles [3, p. 55]—*Deep and Narrow over Broad and Shallow*—describes that LeSS should preferably be adopted in one product group well, instead of applying LeSS in many groups poorly. In case of LeSS Huge, LeSS adoptions should start with one Requirement Area and reach a good state before further scaling.

Since this is a LeSS Huge case, ADD followed the principle described above, and the LeSS adoption started with one Requirement Area.

Figure 16 gives a visual scheme of the Requirement Areas' scope as the LeSS adoption grew. The X-axis represents the cross-functionality of the teams. It shows the level of difficulty. The actions on the right are not a composition of the ones on the left. Developing on a rapid prototyping platform and testing in a car is simpler than integrating and testing the same software on the target platform and in a car. Expanding the scope to multiple ECUs increases the complexity and difficulty a team has to face. Including the co-creation of mobility as a service means working on the whole system, which is impractical, at least today.

The Y-axis shows the product scope, which ranges from a single component to a customer problem.

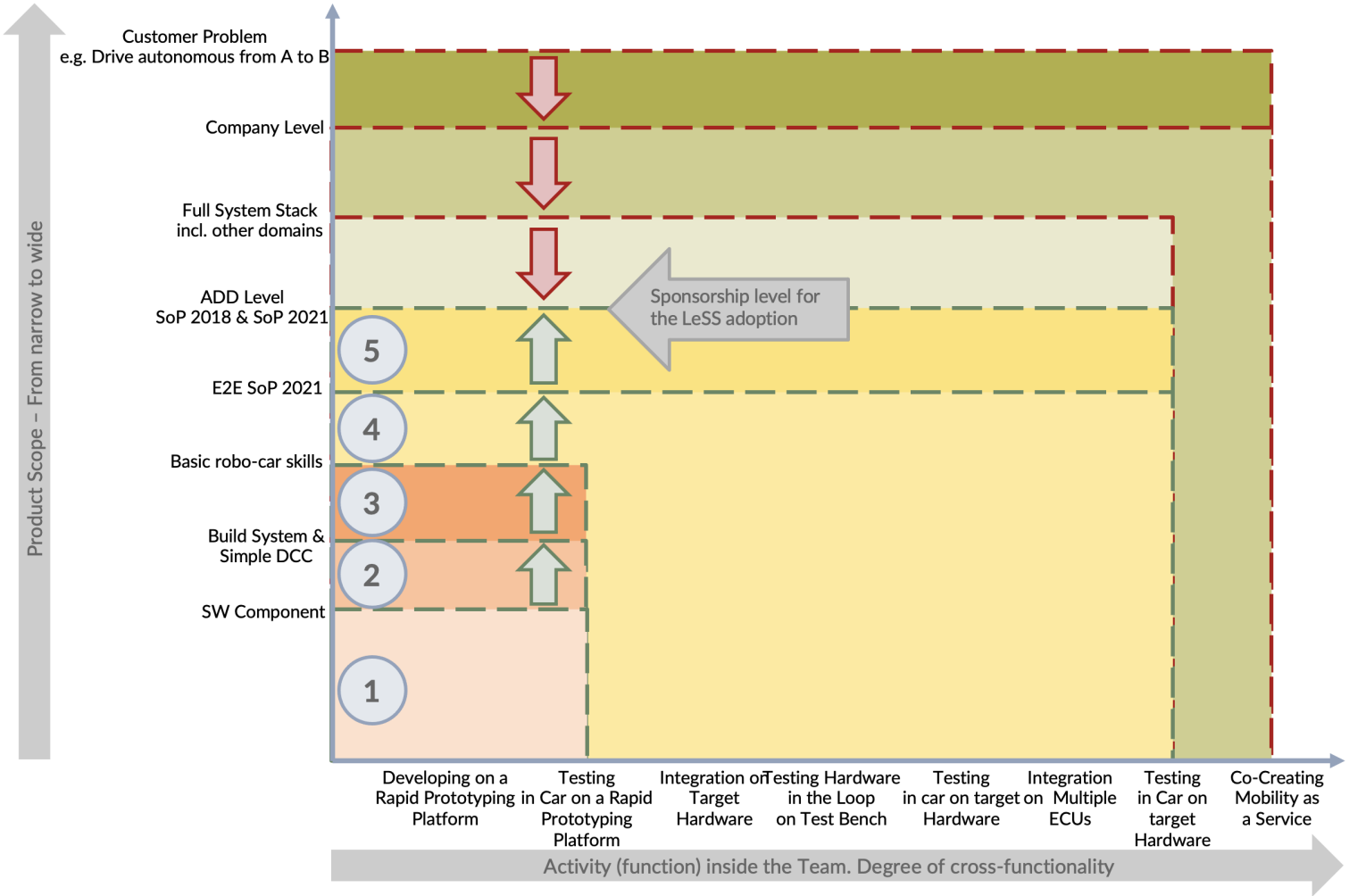


Figure 16: Scope of the LeSS organization over time.

The starting point was component-based development on a rapid prototyping platform (step 1 in Figure 16), far away from the full product scope.

The focus area of the first Requirement Area needed to include the next step towards cross-functional Feature Teams. It was the following:

1. develop a build system sufficient for scaling and adding further Requirement Areas
2. simple Dynamic Cruise Control (DCC) as it involved only a few SW components

Step 1 in Figure 16.

Prerequisites and Constraints

Another principle of the Three Adoption Principles [3, p. 55] is to use volunteers.

> Use volunteers! True volunteering is a powerful way of engaging people's minds and hearts. [3, p. 58]

The intention was to start small with volunteers.

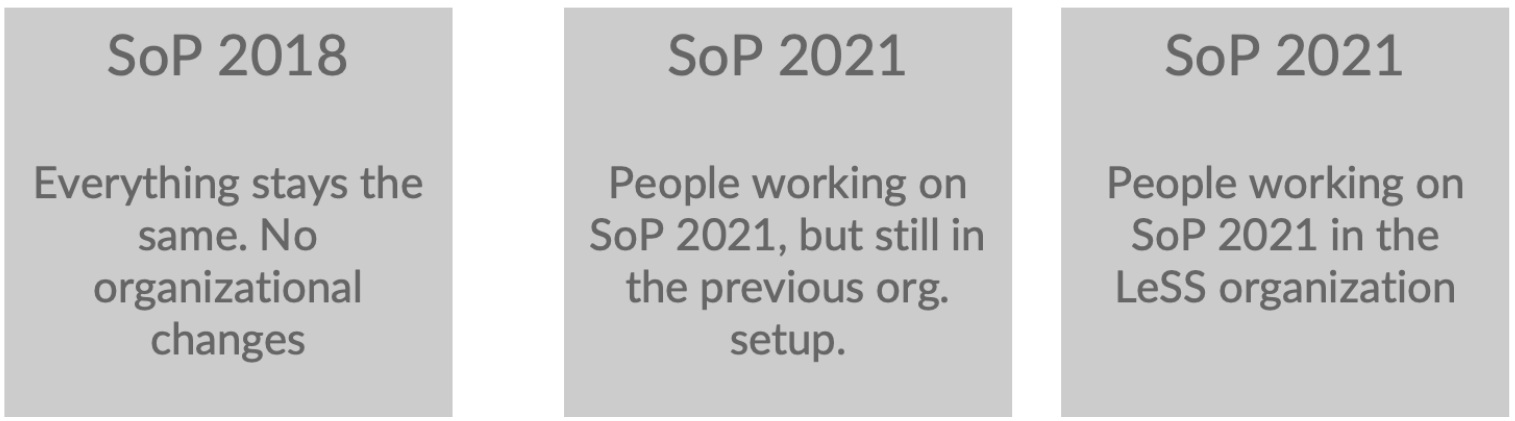
At this point, managers' education consisted of the 1-day introduction to LeSS, Craig's Readings Preparing for LeSS for Executives, and coaching by Mark Bregenzer. Certified LeSS Executive (CLE) courses took place later, together with Certified LeSS Practitioner (CLP) courses of the first Requirement Area's participants.

Setting up the first Requirement Area was constrained as follows.

1. The size of the first Requirement Area should have been around 70 people.
2. The transitioning process to the first Requirement Area had to start at the beginning of August, which meant during the summer school holidays.
3. Functional managers of existing functional/component teams were required to temporarily live a double role when transitioning into the LeSS organization. Those would be (1) a line manager in a LeSS organization and (2) a functional manager in the previous organization. This setup would ensure that employees not yet joining the LeSS organization remain with their functional manager.

Parallel Organization

Constraint #1 and the size of ADD (around 800 people) led us to have a parallel organization as described in the respective guide [3, p. 74]. Looking at Figure 16 makes it clear that the scope of the build system and simple DCC (step 2) would be in the LeSS organization. Everything else (steps 3, 4, and 5) would need to remain in the former organizational structure to ensure stable delivery and stable interfaces to the outside of ADD and BMW Group. Figure 17 visualizes the notion of LeSS, non-LeSS organization, and people working for SoP 2018.



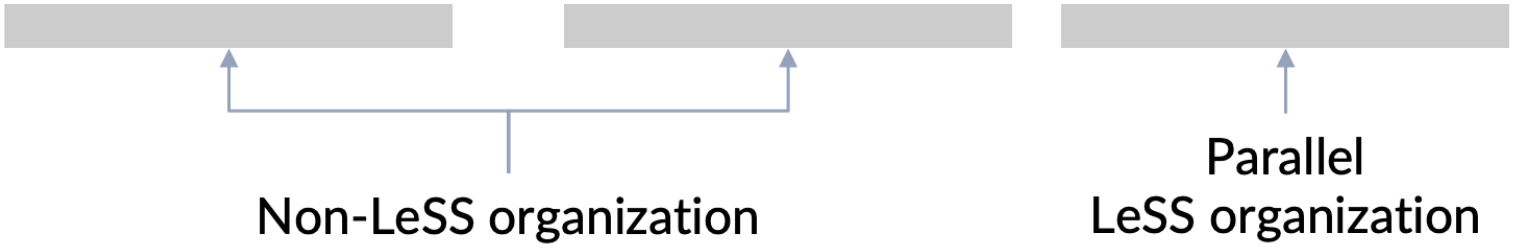


Figure 17: LeSS, non-LeSS organization and the SoP 2018 project.

The parallel organization and the concept of "only volunteers should join the first Requirement Area" resulted in the third constraint.

Fake Volunteers

As mentioned in the section Product Definition, ADD had two major milestones, SoP 2018 and SoP 2021. The people working on SoP 2018 remained in the former organization to ensure the release; they could not join the LeSS organization. Consequently, nearly all product developers, who delivered a car to series production at least once in their life, were not available for the first Requirement Area. This thinking in projects and, as a repercussion, still organizing people around work restricted the pool of available people with the required skills for successful product development in this context. See Organize by Customer Value [3, p. 78] for more information on the topic of organizing people around work vs. work around people.

Further, the beginning of August was also the beginning of the summer school holidays in Bavaria, Germany (see constraint #2). At this time, people with families were on their summer holidays, which reduced the pool of available people even further. The resulting pool of possible volunteers consisted mainly of long-term researchers who never developed a car to the production stage, people who freshly joined ADD, and few experienced key players.

The shortage of available people, combined with the demand to start the LeSS adoption with eight teams in the first Requirement Area, led to forcing people to become "volunteers." The order of actions before the first Requirement Area amplified this effect. First, people from the group of potential volunteers volunteered—in some cases, with a little push. Second, after managers provided a list of "volunteers," they immediately started with CLP classes to educate people on LeSS and give them an idea of what it means to work in a LeSS organization.

Observations during CLP classes showed that some "volunteers" were poorly informed and had little understanding of what it means to work in a LeSS organization. It was the time when the "volunteers" understood what they volunteered for. Some people did not like what they learned. They did not want to be part of the first Requirement Area and became prisoners of the system.

The guide Getting Started [3, p. 59], advises the opposite order—step 0: educate everyone *first*, before volunteering. That wasn't done.

After the Start—Revisit Parallel Organization

Before the LeSS adoption at the BMW Group, managers were usually involved in deciding what the actual work was and how to do it. Further, they conducted individual performance appraisals (IPA) and other line manager related tasks, for example, escalations with vendors and organizational changes. Let's define this type of manager as a traditional manager.

In LeSS, a Product Owner is responsible for the vision of a product and optimizing its impact by prioritizing the Product

Backlog—the *What*. The teams, and *only the teams*, decide the *How* of turning the features or needs into a product. The work of both roles is overlapping intentionally. They should support each other.

The existing paradigm at ADD—having clear lines of responsibility—led to the understanding that the duties or tasks of those roles are mutually exclusive. It became: a Product Owner decides on the What, and teams decide on the How.

Regular Scrum, and LeSS in this regard, don't define the role of a line manager. What should line managers do?

> … the role of management changes significantly from managing the work to creating the conditions for teams to thrive … [1, p. 241].

In other words, their role is to improve the value-delivery capability of the organizational system.

ADD defined the responsibilities for the LeSS organization precisely this way. The Product Owner department was responsible for the *what*. The line managers' roles in the Development department was to improve the delivery capability of the organizational system, and the teams decided on the *how*.

Constraint #3 (see Prerequisites and Constraints) specified that a manager transitioning from a traditional manager to a line manager in the new organization would temporarily need to have a double role. The first would be their previous role as a traditional manager. The second would be their new line manager role. This dual role situation would persist as long as the prior group's subordinates remained in the non-LeSS organization. In other words, a manager who is part of the LeSS organization would still carry out traditional management, deciding on the what and how, in the non-LeSS organization.

The double role setup would violate the distinction of product ownership, line management in LeSS, and traditional management. The concerns about the boundary violation of the roles and organizations led to a rejection of constraint #3. Consequently, the remaining people in the non-LeSS organization became manager free, and no one coordinated the what and the how for them—an unusual situation for those people.

Further, most key players with full system knowledge, if available, transited to the first Requirement Area.

The sum of those circumstances made the rest of the ADD organization strongly dependent on the LeSS organization and they could not deliver without the people in the LeSS organization anymore.

Simultaneously the LeSS organization focused on increasing its delivery capability as a Requirement Area. But, the delivery capability of the entire group working for SoP 2021 decreased, and interfaces to the rest of the BMW Group weakened. Why? Mainly because approximately 250 people in the non-LeSS organization lacked an understandable structure and were lost. Many people stopped doing whatever they were responsible for.

Further, some people of the LeSS organization focused so much on the Requirement Area that they dropped communication and interfaces to the rest of the organization. The pressure for finding a solution increased rapidly, and the lead coach (the first LeSS coach and trainer in this LeSS adoption) became heavily involved in helping to find one. Additional LeSS coaches were engaged in continuing coaching of the first Requirement Area.

The people of the non-LeSS organization needed to work with their colleagues from the LeSS organization. Usually, because someone needed help on topic X, and the topic X expert was in the first Requirement Area. But both groups had significantly different ways of working. "We want to do X in our cross-functional team" vs. "we want to split X across several single-function teams." Explanations of why the people in the LeSS organization wanted to approach and solve things in different ways led to heated discussions and a higher tension between both groups. Both groups did not speak the same language any longer.

Based on this insight, the CLP classes took place independently of the actual transition into the LeSS organization. The entire ADD received CLP classes within one year. However, by the time the people transitioned into a LeSS organization,

their knowledge from CLP classes faded, which created further difficulties in the new Requirement Areas.

Adding More Requirement Areas

Activities such as getting the LeSS organization up and running, creating a solution for the people in the non-LeSS organization, clarifying which legacy code should be part of future development, and many other issues absorbed lots of time and energy. Therefore, progress was rather slow.

There was eagerness for expanding the LeSS organization and starting a second Requirement Area. A few factors motivated it. First, there was a strong desire to become better. It seemed that transitioning people from the almost structureless non-LeSS organization to the LeSS organization would resolve some issues and soothe the pain. Second, the majority of people from the non-LeSS organization who participated in a CLP course were self-motivated to join the LeSS organization as fast as possible. Third, the ADD needed to demonstrate features to their stakeholders and the press. Thus, it was natural that the Senior VP of ADD wanted to see and experience feature increments in the car.

It is worth noting that it is necessary to fulfil some conditions to keep a LeSS organization healthy, especially when adding teams. Those are:

1. A common structure for the Product Backlog of all teams and Requirement Areas
2. A working build system which ensures fast feedback

At this stage, the Product Backlog of the first Requirement Area started mutating to a multi-level, tree-like requirements set, introducing dependencies between items, which obscured the overview of work to be done, the direction to go, and burdened prioritization already for only one Requirement Area. The Product Backlog started to deviate from the most Product Backlog related guides, especially from *Don't "Manage Dependencies" but Minimize Constraints* [3, p. 198] and *Three Levels Max* [3, p. 222].

Before the LeSS adoption two "project groups" had created prototype code for some aspects of autonomous driving; the two sets of experimental code contained lots of duplication and inconsistencies across them. To "speed" things up (another *quick fix*), the entire legacy code from these two bases was merged to the new master branch, with little automated testing, leaving the feature and code coverage at an insufficient level (for more details see Merging Repositories). Consequently, shared code ownership became significantly more difficult. The build system and infrastructure were not ready for this structure and load. The whole system became slow, and feedback times increased significantly. As a consequence, trunk-based development with fast feedback cycles became impossible (more on this topic in the Technology View chapter).

The second Requirement Area kicked off with a broken backlog and build system. And the scope for the LeSS organization expanded to *all* software components and sensors, yet still being developed on only the rapid prototyping platform (step 3 in Figure 16).

To expand from one Requirement Area to many while sustaining transparency about item priorities in the Product Backlog, it must have a common structure for all Requirement Areas. It enables the PO to better grasp what's going on, and to prioritize and change teams between Requirement Areas—a quality of *organizational adaptiveness*. But, the tool, which was used as an electronic Product Backlog, and the people using it, created obstacles. Combined with the obscure multi-level, tree-like requirements set in the Product Backlog, its structures between Requirement Areas (existing and planned) began to diverge at about this time—a *local optimization*. See also *Guide: Tools for Large Product Backlogs* [3, p. 210].

Unfortunately, even with these clear weaknesses in place, adding new Requirement Areas continued for the next few months (see Figure 18).

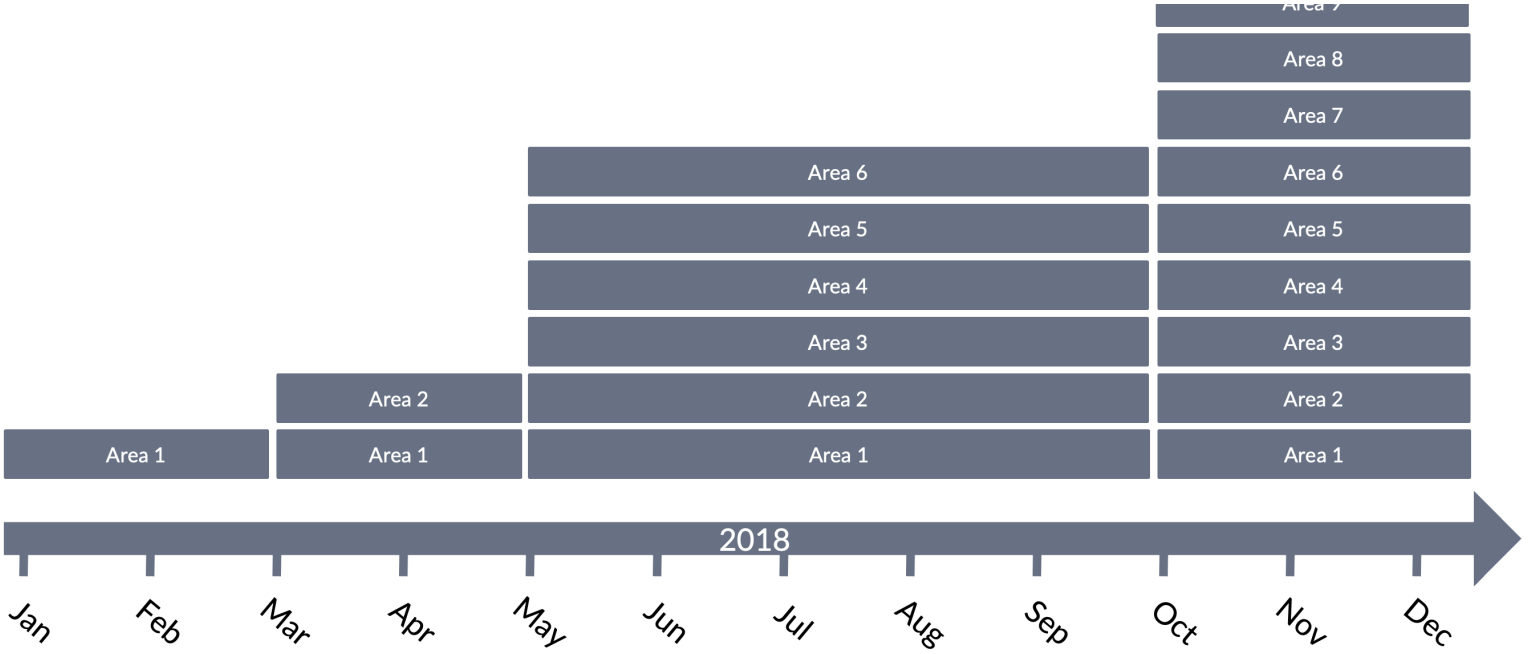


Figure 18: Requirement Area ramp up over time.

With each step, the scope of the LeSS organization expanded. Adding four Requirement Areas expanded the scope to step 4 represented in Figure 16. With all nine Requirement Areas, the scope reached its maximum, the entire context of ADD (step 5 in Figure 16), yet myriad weaknesses underlay this too-rapid expansion.

Numerous experiments were necessary to find a meaningful scope for each Requirement Area. It took several restructurings, inspect & adapt cycles. During those restructurings, the flexibility of a LeSS structured organization became visible. Since Requirement Areas were *not* part of an organizational chart or unit, the restructuring process was quick. The fastest restructuring of Requirement Areas (inception until complete implementation), happened within one week.

That was the good news: an adaptive organization. But the bad news was that a key driver for this frequent restructuring was expanding too fast.

Although a LeSS Huge organizational structure allows such flexibility, it is intended to accommodate changing priorities in the customer-centric view, and it comes with some *switching costs*. Due to interrupting evolved inter-team relationships and established collaboration within one Requirement Area, and the non-trivial domain learning required for teams moving to a new area, *frequent* restructuring of Requirement Areas has issues.

Retrospective On The Timeline View

> Adjusting organizational structure is relatively easy, but changing mindset takes time, discussion, introspection, and learning. [1, p. 229]

The BMW Group's LeSS adoption can be visualized with the help of the Satir change model. Figure 19 shows the journey to the current state and how it could look like in the future.

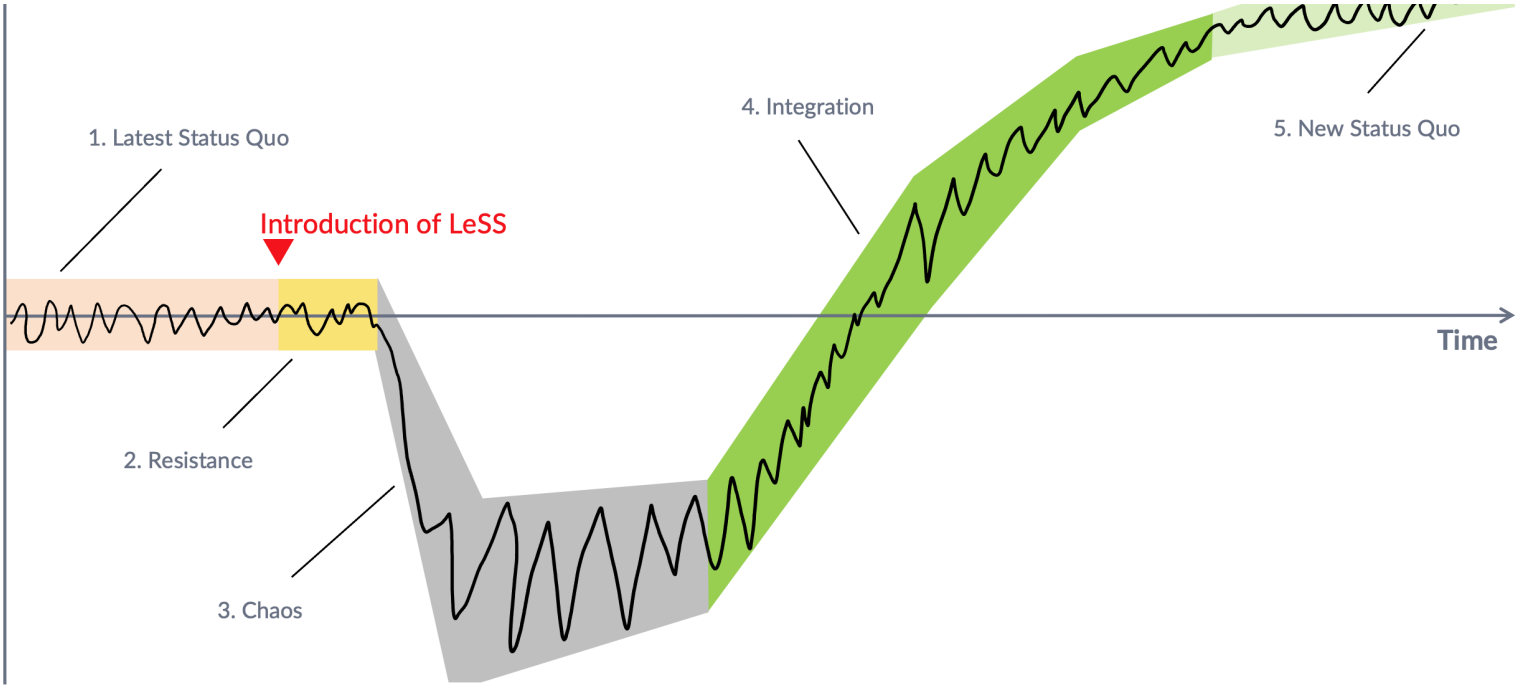


Figure 19: Journey of BMW Group's LeSS adoption. Based on the Satir change model.

The journey consists of 5 phases. This retrospective view elaborates on the Chaos phase only.

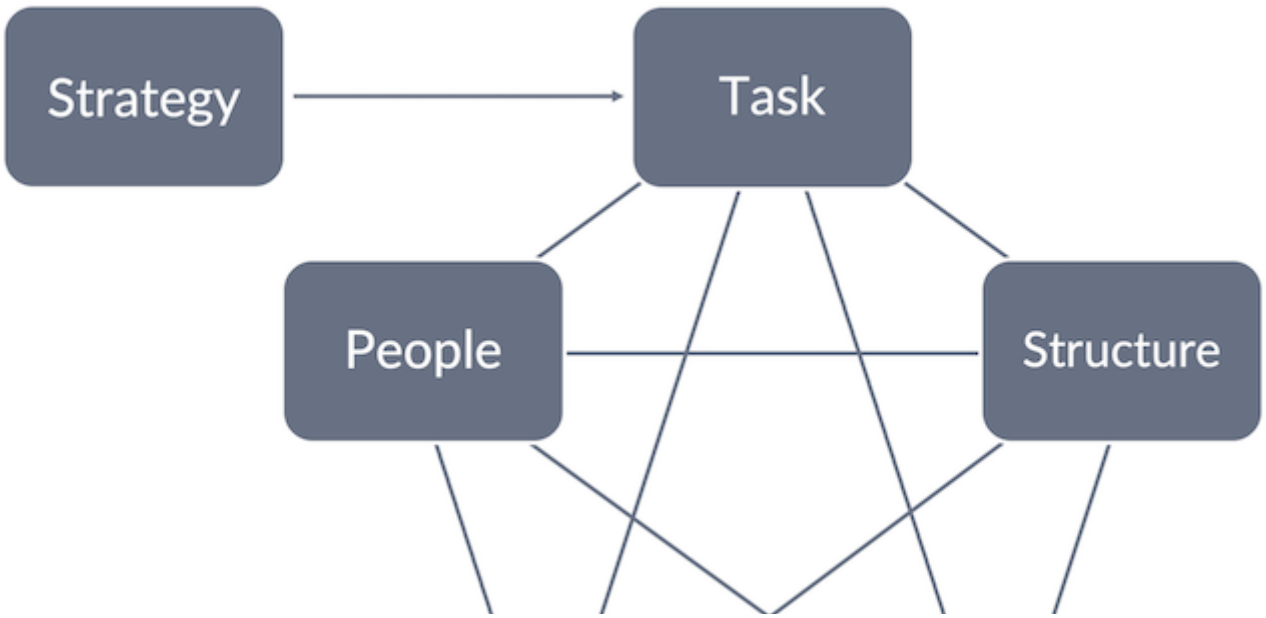
After introducing LeSS, ADD entered this phase quickly! Cognitive biases and the system all humans are in influence our mental models and, therefore, our behavior. Craft mistakes and active sabotage were its effects, leading to the situations described in this report.

The question is: Could the magnitude of the chaos have been minimized or even avoided?

Probably, it could have! And so perhaps you the reader can benefit from some lessons learned. This retrospective view, covering two years after the original steps towards LeSS, exposes causes of the painful dynamics and proposes ways to prevent or minimize them.

The Product Development System

Let's elaborate on what the product development system is, using Jay Galbraith's organizational design framework—the Star Model™ (see Figure 20).



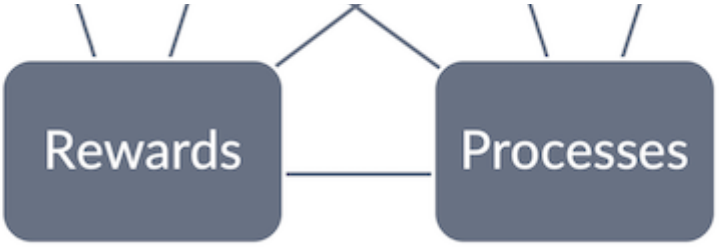


Figure 20: Jay Galbraith's organizational design framework, called the Star Model. This version is from 1998.

Craig Larman and Bas Vodde observed in their book *Scaling Lean & Agile Development* that a Scrum or LeSS adoption directly alters the *processes* and *structural* elements. In ADD's case, precisely those elements were focused on when preparing for the LeSS adoption.

Structure and processes are only *two* parts of an organization's design, and often too many efforts are spent on just them and too little on the other elements.

> Structure is usually overemphasized because it affects status and power... [14, p. 4]

The Star Model elements are highly interwoven and have influential forces on one another, which *together* influence the behavior, culture, and performance of the organization. Therefore, alignment between *all elements is crucial* for an organization to be effective. Otherwise, the organizational capability *decreases*. Galbraith put it this way:

> For an organization to be effective, *all the policies* (elements in the Star Model™) must be aligned and interacting harmoniously with one another. [14, p. 5, explanation in parenthesis added, emphasis added]

This retrospective analysis is structured using the Star Model elements.

Strategy & Task

> Effective team self-management is impossible unless someone in authority sets the direction for the team's work. [12, p. 62]

Despite a LeSS introduction and decision to manage *product* development as product development, the reality felt like *project* development.

The BMW Group had (and still has) a Product Management department, which accommodates people who plan and manage the entire life-cycle, from idea to development to maintenance, of all vehicles that the BMW Group offers. This department sets up *modular systems contracts* with other departments that develop car parts. Such a contract usually covers a set of different vehicles, their release dates, general scope, budget, and aims at high reuse of the components/systems the departments develop.

Before the LeSS adoption, such a modular system contract (an actual document) was signed between the ADD and Product Management departments.

The traditional automotive industry still does long-delayed integration rather than frequent. Therefore, there was (and still is) a BMW Group-wide general car project timeline, which defined fixed integration steps and other milestones across all involved departments on the way to the start of production so that they can synchronize their work on a slow cycle.

Figure 21 illustrates the setup between the modular system contract, the general car project timeline, and ADD.

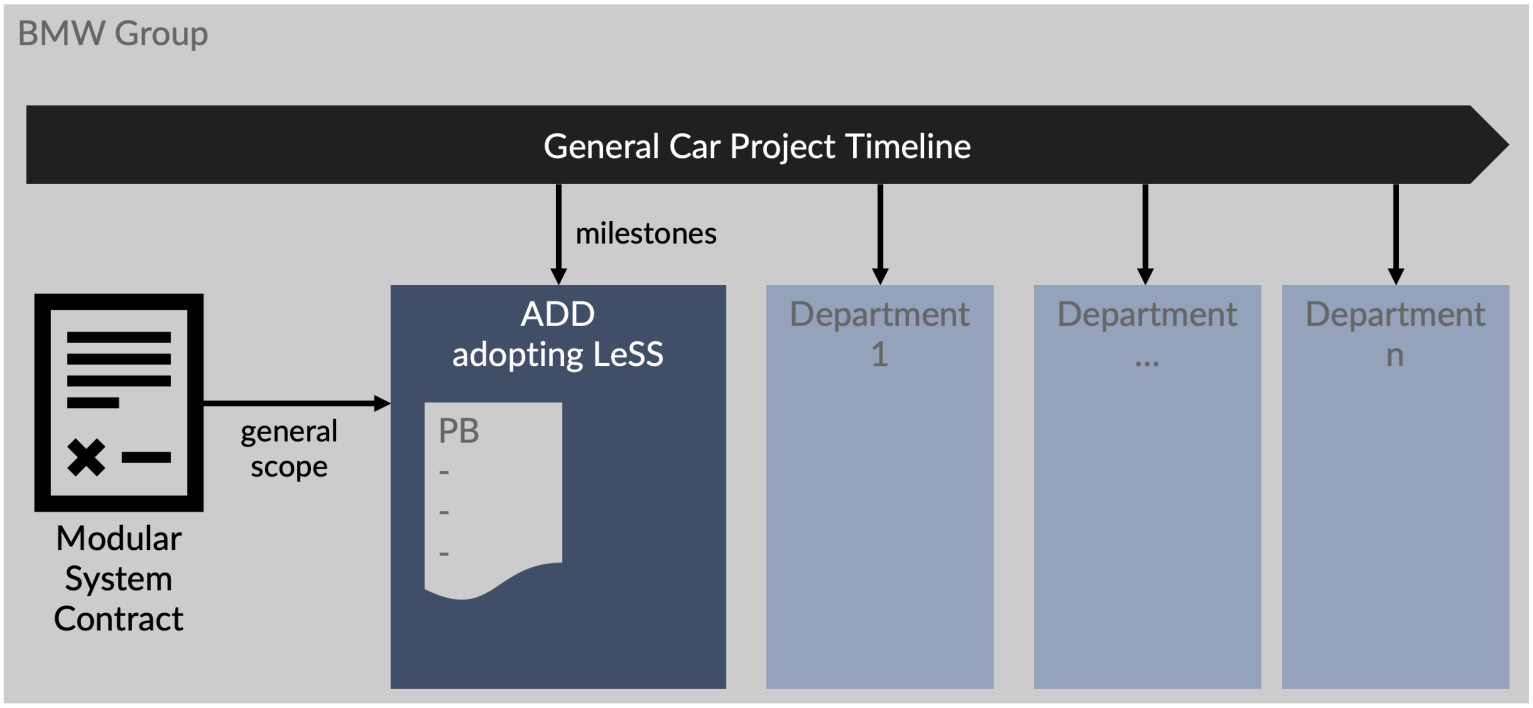


Figure 21: ADD-external constraints which should influence the Product Backlog.

Over decades managers conveyed the message that "we need to deliver *everything* and on time," meaning the contract's scope, by the release date agreed upon. The message turned into a mantra, spread and believed in by many people, not just managers. Consequently, the desire to deliver everything was high, and the message continued to spread. Further, this ethos arose in the context of creating *electro-mechanical components* such as a braking system; which are *infinitely* less complex, less variable, and less research-oriented than the profoundly hard job of creating AD software.

Interestingly, the people conveying this message ignored the evidence that, at least in the last decade, the product group *never* delivered "everything" on one given deadline. There usually was an intensive work mode (like a task force), where someone deprioritized less important topics to focus on the mission-critical ones.

Further, the scope described in the modular system contract was, to a degree, negotiable. What was that degree? All the features a customer could experience in the latest car model must also be available in a new model. Therefore, the scope of existing features was a must-have. However, the content of new features was negotiable.

Despite the evidence, those circumstances diminished the acceptance of reducing-the-scope discussions (at least at an early stage), leading to no prioritization because "everything was crucial." The resulting behavior was "we need everything." Wishful thinking! Of course, if everything is equally important, everything is equally unimportant. Only one year before the release, deprioritization started—meaning re-negotiations of the scope and date. Such late re-negotiations were—and still are—common and in line with other BMW Group projects; therefore, they were expected.

Another aspect led to an unprioritized so-called "Product Backlog" (so-called since, per definition, a real Product Backlog would be prioritized, providing a clear direction). Most not-really-APOs "APOs" were project managers who had also previously been developers.

The first Requirement Area started with such "APOs" acting as a fake PO because it was only one Requirement Area, and

the person who later became the PO was not fully available. Sometime after starting the first Requirement Area, both "APOs" rejected coaching, especially on setting up a real Product Backlog. Why?

To start with, the learnings emerging during the initial coaching sessions were undoubtedly uncomfortable. For example, accepting that the idea of *managing the product complexity* by splitting it into small parts is an illusion and that instead, empirical control, learning from product experience feedback, and prioritization are a better approach.

And some "APOs" in high power positions refused LeSS coaching from people they saw under their status level—basically all coaches we engaged.

The resulting lack of PO/APO and Product Backlog competence led to a "Product Backlog" full of technical tasks on multiple abstraction layers and many dependencies between them, which was the main impediment for the PO to prioritize the Backlog.

Key point: Most "APOs" could not keep the whole product focus and derive valuable items for the next Sprints. Instead, they tried to split everything into small parts—a decades-of-practice habit and a fear response of forgetting something.

Those circumstances were very convenient for the teams because technical tasks narrowed their focus to just one or two components but didn't motivate them to learn the customer language, nor to learn across a broader set of components and skills to increase their learning and *adaptiveness*. In consequence, the so-called "Product Backlog" consisted mainly of technical tasks instead of customer-centric items. The result was two other anti-patterns. (1) Re-prioritization on the product level became difficult—in fact, close to impossible—because technical tasks naturally depended more on each other. And (2) collaboration and coordination opportunities between teams when finding technical solutions for customer-centric problems were hard to find. Why? Because technical tasks typically reflect only a small part of the whole system, for example, one component, but customer-centric items usually span multiple system elements.

Both anti-patterns led to overloaded so-called "APOs." Some of them acted as single-team "POs," prioritizing specific team backlogs in their Requirement Area, which reinforced and amplified the downward spiral from a product-requirement to technical-task perspective.

The result? Lack of whole-product focus and prioritization, leading to high busyness and *output* of completed technical tasks, but very low output of completed customer features, and thus no useful *outcome*. This typhoon of technical tasks made it impossible for the PO to have a meaningful whole-product overview, to order the so-called "Product Backlog," which disempowered him and made him dependent on the technically involved "APOs"—he had to believe what the "APOs" told him.

Why didn't the PO clean up this mess? Why didn't he enforce a real Product Backlog enabling him to order it? One cause is the lack of time for the job as PO. The person playing the PO role had many other duties; for example, VP of a department, project manager of a project external but adjacent to ADD, and most importantly, he was occupied with the SoP 2018 release. And there are very likely other reasons that remain invisible to us.

It begs the questions of why a "free" PO was not chosen and why we thought an overburdened person could effectively learn and do a complicated new approach?

No-one, at least to my knowledge (Konstantin here), questioned whether or not this person should become PO. He was considered a perfect fit for the PO role by people on all hierarchy levels. Why? Because before the LeSS adoption, he was the VP of the customer offerings department and re-prioritization was in his job's nature.

Having time to live out the PO role was not considered an issue because he could have delegated other activities to his staff and free himself up.

Therefore, the better questions are: Why did he become overburdened, and why didn't he receive PO coaching?

To start with, the PO lapsed into a traditional management paradigm. He delegated most of the PO tasks and trusted that his staff would execute them properly.

Further, he knew about the problem of technical tasks "Product Backlog." But, he decided to support his so-called "APOs" mainly because the "APOs" saw the lead coach as someone who never delivered a car to production, which degraded the coach in the "APO's" and PO's eyes. Therefore, the PO trusted his "APOs" more.

> … the failure to establish a compelling direction runs two significant risks: that team members will pursue whatever purposes they *personally prefer*, but *without any common focus*; or that they gradually will fade into the woodwork … [12, p. 80, emphasis added]

How could this situation be resolved?

> The key is to identify who has the legitimate authority for direction setting and then to make sure that that person or group exercises it *competently*, *convincingly*, and *without apology*. Team performance greatly depends on how well this is done. [12, p. 63, emphasis added]

A better approach would have been to free up the PO of other duties, which have little to do with product ownership. Further, replace the so-called APOs with real ones who focus on the product instead of technical solutions.

Further, de-escalation and mediation sessions would probably have been helpful to re-establish PO and APO coaching.

Moreover, a Product Backlog (PB) transformation from SW component-centric and technical tasks to customer-centric features would enable the *PB's prioritization* and allow teams to solve real customer problems.

Whole product focus is crucial! Inviting users, paying customers, and subject-matter experts to the Product Backlog Refinement (PBR) sessions would have helped establish a *customer* language, gain *whole product focus*, and deliver the ADD product incrementally with evolving customer features rather than completing technical tasks.

How to involve customers in PBR and comply with information-protection policies? It is worth realizing that many of BMW Group employees are also *customers*. They drive BMW cars and experience the product themselves. The only constraint would be to involve those who don't understand the technical details and only speak the customer language—for example, people from HR, legal, or procurement.

Structure

> The job of managers is to build an environment in which teams continuously deliver and continuously improve. [3, p. 69]
> Managers' role is to improve the product development system by practicing Go See, encouraging Stop & Fix, and "experiments over conformance." [3, p. 115]

The starting structure allowed a high degree of organizational adaptiveness, which we experienced when reforming Requirement Areas. Since Requirement Areas were *not* part of an organizational chart or unit, the restructuring process was quick.

The ADD used this benefit to frequently change Requirement Areas, mainly caused by a too-fast expansion of the LeSS organization (see also Adding More Requirement Areas). Why did the LeSS organization expand too fast?

One cause lies in the BMW Group's traditional career paths, where good hands-on engineers at the top of their technical

career became mediocre managers. Consequently, the *sunshine to bad-weather managers* ratio was high, decreasing the managers' capability to structure and manage the non-LeSS organization, subsequently leading it to an unhealthy state, which became one strong force to bring the people into the LeSS organization faster.

Another cause—likely related to the previous—was the non-LeSS group's inability to work independently from the LeSS organization due to a lack of experts, who mainly were in the SoP 2018 project and the LeSS organization. This situation is a consequence of thinking in and organizing people around projects instead of products.

Despite the thoughtful groundwork of organizational structure and striving for self-managing teams and collaboration in communities of practice, people perceived self-management, particularly in communities, as ineffective. This was because of a vast amount of unskillful decisions and weak decision-making, due in part to the lack of experience and ability in decision-making amongst developers (since this was previously done by people in specialist roles and managers) and also in part to the lack of experience and skill amongst the Scrum Masters, who did not effectively coach the teams or communities in participatory decision-making protocols.

As a *quick fix*, for this and related reasons, senior managers revived C-4 level managers and specialist roles around two years after starting the first Requirement Area. Further, the organizational structure fossilized such that Requirement Areas became official organizational units, which leads to *rigid hard-to-eliminate* groups instead of adaptive birth/grow /shrink/die Requirement Areas, which are essential within LeSS Huge. Further, official organizational groups reinforce the silo problems. Unfortunately, this insight was either never present or forgotten at ADD, which led to ignoring the guide LeSS Huge Organization.

> Avoid having the Requirements Areas be equal to the organizational structure as it leads to them being difficult to change. [3, p. 110]

Let's elaborate on why many decisions were unskillful, and decision-making was ineffective.

When changing from manager-led to self-managing teams and communities, managers' and sometimes Scrum Masters' thinking was, "Either the teams/communities are self-managing, or I need to step in, breaking the self-management."

Key point: This was a *false dichotomy*. Instead, it should have been: "I need to *help them become self-managing*."

Driven by the fear of breaking self-management, managers left the teams and communities mostly to themselves. Since they had to make decisions, the most inadequate and naive decision protocol became their default, *majority voting*, another *quick fix* not seen, and even worse, supported by junior Scrum Masters. The experienced people were a small minority and were outvoted most times. Therefore, most decisions were unskillful and ineffective.

Key point: We failed to establish a skill hierarchy (without specialist roles) and introduce adequate decision-making protocols, such as a quorum of skilled and experienced people makes decisions.

The situation became similar to the one described by Jo Freeman in her article *The Tyranny of Structurelessness* [15].

> … the movement generates much motion and few results. Unfortunately, the consequences of all this motion are not as innocuous as the results' and their victim is the movement itself. [15]

How could a specialist-role-free skill hierarchy look like?

First, make transparent who has beginner, intermediate, advanced, or expert skills and their fields. Second, ensure that people with advanced and expert skills form a quorum for decision-making and introduce participatory decision-making protocols, such as the Decider/Resolution from The Core Protocols [19].

Third, make the differences between the skill levels transparent so that people know how to advance. Fourth, help people gaining advanced and expert skills. Moving from beginner to intermediate can usually be done by acquiring knowledge. Moving from intermediate to advanced and then to an expert level takes time, where the knowledge gets enriched by the experience.

Processes

Do Continuous Improvement Towards Perfection... *Continuously*

We wanted to follow the Deep and Narrow over Broad and Shallow [3, p. 55] adoption principle and adopt LeSS step-by-step, one Requirement Area slowly after another. The contract engaging coaches defined a fixed-price package of bringing 70 people into the LeSS organization at a time.

The emphasis on complying with the contract was strong, especially in the beginning. Otherwise, there would be less "value" for the same money—a *contract game*. This dynamic led to the... *prerequisite of having 70 people in the first Requirement Area*! Combined with the demand of starting the LeSS adoption at the beginning of summer holidays, and therefore little availability of people, this led to *forced* "volunteering" (see also Prerequisites and Constraints and Fake Volunteers). Thus, most Requirement Areas started with this number of people.

The contract also contained a project plan—a Gantt chart describing when the LeSS adoption starts and ends. This project plan, based on ignorance of skillful change and improvement, aimed to comply with the purchasing processes and BMW Groups' policies. Behind the scenes, of course, there was no intention by those understanding adaptiveness for it to be used as an actual project plan.

But of course Murphy's law came true: senior managers discovered this project plan, and then ADD followed it, mainly because it indicated an end date of a *continuous* change. Consequently, and this plan is only one cause, the LeSS organization expanded mostly according to plan rather than according to what made sense! See also Adding More Requirement Areas and Structure.

Key point: Continuous Improvement Towards Perfection was seen and run as a project instead of *continuously*.

Avoid Making Departments or Individuals Responsible For a Special Thing

Whenever an existing process needed adaptation or compliance with industry standards required a new one, most of the time the PO team and the development department delegated these tasks and withdrew themselves (see Figure 15). Why?

One cause is that people were used to working this way. Another is that the important idea of those who use and live the processes *make* the processes was, to a large extent, not lived.

Key Point: In LeSS, *teams* are responsible for their processes. It is a direct consequence of having self-managing teams. See *Guide: Understand Taylor and Fayol* [3, p. 115].

One of the differences between self-managing and manager-led teams is the responsibility for their processes. Self-managing teams *own* their processes. Since this understanding was lacking at the BMW Group and due to organizational size and dependencies external to ADD, most people *rented* the processes rather than owned them.

Let's take a look at the reasons why people did not take responsibility for their processes.

During the first two years of the LeSS adoption, people had doubts about their ability to align with LeSS principles and

rules when changing existing or designing new processes. To resolve this situation, Scrum Masters coached LeSS principles and helped to create or improve processes. It is a healthy dynamic.

During the preparation phase, before the LeSS adoption, this dynamic was foreseen. To ensure alignment with LeSS and industry standards, the *Competence and Coaching* department, which consisted of Scrum Masters and industry-standardization experts, became the official body responsible for processes.

Why did the Scrum Masters allow themselves to get involved in something so obvious that they should not? Due to little Scrum experience in the entire ADD, there was a deeply engraved understanding that Scrum is somewhat a process. As a corollary, there were *no masters of Scrum*. Instead, there were junior Scrum Masters who did not foresee the dysfunctions the "responsible for processes" label would create.

The official "responsible for processes" label created a mental barrier in people's minds. People sought approval from the Competence and Coaching department for any changes in any process. After some time, this turned into handing problems over to the Competence and Coaching department and expecting them to drive their resolution.

As a *quick fix*, the Competence and Coaching department was split into two—(1) A unit of industry-standardization experts and (2) a homogeneous Scrum Master organizational unit. Two functional groups, each with their single-function line manager. This was an example of the "default organizational problem-solving technique. (1) Discover a problem—the blah-blah problem. (2) Create new role—the blah-blah manager. (3) Assign problem to new role." [3, p. 120]

In the long run, one negative and unforeseen side-effect was that the industry-standardization experts unit ended up creating various unskillful processes for the teams since they had little hands-on experience nor insights. In retrospect, with more focus on the experiment *Avoid…Adoption with top-down management support* [2, p. 374], we might have been able to anticipate this since it warns of this very problem.

In conclusion, making an organizational unit responsible for processes is counterproductive for human workers in about any context. It is a manifestation of the very basic tayloristic approach of separating work between people who plan and people who mindlessly execute the plan.

Key Point: In fact, "responsible for anything" labels create bottlenecks, delays, hand-offs, various other wasteful dynamics, and, most important, the ownership problem. That is when the psychological connection between the role or group responsible for X and other people owning X gets broken.

The Competence and Coaching department received the "responsible for processes" label for people from other departments to know whom to seek advice from when changing processes. However, this label created an unwanted dynamic.

A more effective means to the same end would be to (1) allow the industry-standardization experts to join teams to gain hands-on experience, and (2) have regular team members be centrally involved in creating the processes.

Rewards

During preparation for the LeSS adoption, coaches strongly emphasized the importance of changing the rewards system (including the salary model) to support the other elements of the product development system.

Specifically, subjective *manager-driven* promotions and manager-based *individual* performance appraisals (IPAs) needed significant changes to support *self-managing* teams. Why?

Manager-driven promotions drive the desire for manager-based IPAs, yet the manager (appraiser) virtually never has the insightful information to skillfully appraise. And then there are biases—conscious and otherwise. Combined with long cycle times (in this case 12 months!) between IPAs, the feedback becomes *superficial* and *unrelated* to the individual's specific behavior. Hence, the feedback quality for the individual's betterment is highly questionable.

Further, since manager-driven promotions are *subjective*, people start to believe that promotions are less related to performance but more related to having the "right" relationships and an ingratiating personality. Jeffrey Pfeffer described this dynamic in his worth-reading *Harvard Business Review* article *Six Dangerous Myths About Pay*.

This leads to individualism, heroism, and other dysfunctions in an intended self-managing *teamwork* environment.

At ADD, C-3 level managers, together with the HR department, had the aspiration to change the existing IPA to support teamwork and make the rules for individual career advancements transparent. But they failed.

This situation wasn't a problem until…

…The next cycle of IPAs. Since nothing changed, people still expected their line managers to conduct their IPA. But! …The elimination of the C-4 level managers increased the span of each C-3 line manager to about 60 employees, who now had many IPAs while at the same time having even less sight and insight.

This dynamic made the already-subjective IPAs even more subjective. Later, an employee satisfaction survey confirmed people's frustration with this situation and their managers. Urgent improvement was necessary.

As a *quick fix* senior managers revived managers on the C-4 level to decrease the span and generate a false feeling of objectiveness in the next IPA cycle. Back to the status quo!

Conclusion

When introducing a significant change, such as a LeSS adoption, and changing only the structure and processes, but not reward policies and other elements, the change will fail. It's a system!

> The Star Model™ suggests that the reward system must be congruent with the structure and processes to influence the strategic direction. Reward systems are effective only when they form a consistent package in combination with the other design choices (elements in the Star Model™). [14, p. 4, explanation in parenthesis added]

If upper managers don't exercise systems thinking, then the system will "revert to mean" simply because everybody knows it. Only changing parts of the system and having high expectations causes pain. It forms the opinion, "We have tried X, it does not work," and leads to reactive responses, which usually result in *quick-fix reactions*.

What could be an alternative?

A shared objective is necessary before evaluating alternatives. If the goal is employees' short-term compliance, then behavioral manipulation is probably the best approach. In this case, it is senseless to work towards self-managing teams because both aims contradict each other.

The next paragraphs assume a shared agreement on the goal of having self-managing teams. To elaborate alternatives, the practices of *individual performance appraisal* and *promotions* need some deconstruction.

Individual performance appraisal:

The common two purposes of a performance appraisal are:

1. feedback for betterment (improvement and development, e.g., adding new strengths or adding to existing strengths), and
2. input for promotion.

Feedback for betterment is supposed to be a two-way conversation, conducted frequently rather than annually, and it is utterly divorced from promotions.

> Providing feedback that employees can use to do a better job ought never to be confused or combined with controlling them by offering (or withholding) rewards. [17, p. 185]
>
> It seems foolish to have a manager serving in the self-conflicting role as a counselor (helping a man to improve his performance) when, at the same time, he is presiding as a judge over the same employee's salary… [18]

In a Scrum environment, the team members and their Scrum Master—who are working closely together and understand in detail each other's work—have the opportunity to give feedback to each other *every* Sprint Retrospective. Since this is standard practice, the value of this purpose in IPAs vanishes.

What is left are promotions.

Promotion (and Rewards): a decision (by another party) for someone to be categorized in a new role classification that is considered "higher" and a desirable step. It *usually* includes a raise in *one* of the following: salary, power, visibility, prestige, influence, decision authority, recognition, etc.

What is usually interesting to people is not a promotion per se, but what comes with it, and promotions usually offer different types of *rewards*. Frederick Herzberg categorized some of them in his *Harvard Business Review* article "One More Time: How Do You Motivate Employees?" into two categories: (1) *hygiene factors* and (2) *intrinsic motivators* [16]. According to Herzberg, a bad implementation of the hygiene factors leads to job dissatisfaction. However, getting those right does not lead to job satisfaction. Instead, it leads to *no job dissatisfaction*. Intrinsic motivators are the ones which lead to job satisfaction.

In this context, *salary is just a hygiene factor*. Therefore, it is useful to treat it as *compensation*, not as a performance contingent reward. With that in mind, the entire practice of manager-driven and -influenced salary adjustments become ineffective and can be removed.

> Do your best to make sure they (employees) don't feel exploited. Then do *everything in your power to help them put money out of their minds*. [17, p. 182, explanation added in parenthesis]

Note that the *non-monetary* rewards (e.g., decision authority, recognition) associated with promotions are more likely to act as intrinsic motivators.

Dr. Richard Hackman (who was a leading expert on teams, at Harvard university) mentions three conditions for rewards to work. (1) "team members must *understand* what it is that is wanted and rewarded." (2) "There must be trustworthy *indicators* of the degree to which the desired outcome actually have been achieved." (3) "… members must perceive that they have *leverage* on the attainment of those outcomes, that their collective behavior directly shapes the outcomes that trigger the rewards." [12, p. 138-139].

Formulaic-criteria promotions could help to achieve all three points. They would also indicate the Product Developer (PD) level.

What could be the formulaic criteria? In a LeSS context, the criteria must help increase the adaptiveness of the organizational system. For example, to go from PD_n to PD_n+1, the employee has to indicate level X in *four* different skill areas, showing they are adaptive. This way, formulaic-criteria promotions become more objective, transparent, and clear to everybody.

How to assess the employees then? As in any assessment, the assessor must be more experienced than the person being

assessed. Therefore, it could be done by internal widely-recognized experts or an expert panel, or by an external body, such as a group that evaluates and certifies expertise in a skill. Further, feedback from peers could also be used.

What to do if the HR policies remain unchanged?

Just like in ADD's case, managers couldn't change HR policies or remove the IPA. The book *Scaling Lean and Agile Development* contains two experiments in this regard.

1. *Try…Discuss with your team how to do appraisals* [1, p. 275]
2. *Try…Fill in the forms* [1, p. 275]

In step one, managers and the teams need to discuss the dysfunctions manager-based individual performance appraisals and manager-driven promotions create. After they reached a shared understanding and understood that the process is unskillful and wasteful, they must *agree* on how to handle the unchanged HR policies in the future.

In step two, after recognizing the process is a waste, they do the process with the *least effort and attention* and then go quickly back to the useful work: *shipping the product*.

People

Basic systems thinking—and the Star Model's interwoven nature (see Figure 20)—shows that people need a holistic appropriate organizational design to succeed.

Key point: Peoples' attitudes and the other Star Model elements need to fit together for the organization to be effective.

The team members were coming from a traditional organization, manager-led and organized in single-function groups and component teams. The LeSS adoption required people who *want* to work cross-component in *shared code* and cross-functional solving customer problems—a quite different attitude.

Many people had difficulties working this way, regardless of whether they wanted this or not. Over time, most teams reverted to a component-team-like setup. Let's explore the driving forces behind this phenomenon.

One strong cause was the technical-tasks list of work, incorrectly called a "Product Backlog". See Strategy & Tasks.

Another cause was the peoples' desire to narrow their focus to fewer, ideally one software component. There were at least two reasons for that:

Reason one:

Before the LeSS adoption, ADD estimated (based on software components) the additional number of people required to develop AD. How much more complex will each software component become, and how many more people will we need to cope with that? This implies that the dominant mental model about organizational structure was based on components instead of customer-centric features.

Hiring efforts before the LeSS adoption attracted many new people with a component-thinking attitude since people with specific expertise in specific algorithmic fields were looked for. Further, most new-hires came from universities or their first jobs and had little to no product development experience, especially working in shared code.

With the LeSS introduction, the *initial* customer-centric end-to-end Product Backlog items rarely covered peoples' existing expertise. The fear of losing their competencies and the illusion of losing their market value made the people demand to work only on "items" (i.e., single-specialist tasks) covering their specialization, especially since this was why ADD hired them in the first place.

The traditional-management need of utilizing people in their specialization, combined with some manager's unwillingness to help their subordinates resolve their issues, led to accepting the peoples' demands and creating work (tasks rather than customer features) designed for their single specialty—another *quick fix*.

A more useful approach would have been to take only real volunteers into the LeSS organization. Further, help each person individually resolve the issues they had, mainly a widespread false dichotomy of "Either I do what I like and good at (single specialization), or I'm a generic feature team member." In the worst case of not wanting to work in such an environment, managers should have assisted individuals in finding another position.

Reason two:

The learnings from Certified LeSS Practitioner courses motivated many people to work cross-component and cross-functional. However, the lack of software craftsmanship before the LeSS adoption accumulated so much technical debt that it blocked simultaneous working in a shared code manner.

The problem was further exacerbated by having so many new teams involved, and more or less at the same time. It became impossible to adopt software craftsmanship with so many people involved, while dealing with dirty code.

For example, low test coverage and high coupling between components left people in the dark about whether their changes broke another part of the system. People had to rely on component experts' judgments on whether their changes would break something. Consequently, people were reluctant when changing the parts that they were not experts in.

Therefore, people could not handle the technical stack's wide scope effectively and requested a focus reduction. These effects became evident, especially as the LeSS organization expanded—another effect of too-fast expansion.

Key point: Large technical debt, too-rapid expansion of the LeSS organization, and little software craftsmanship skills lead over time to component teams—a reinforcing downward spiral.

At that time, technical coaching was available en mas at ADD. Unfortunately, many teams did not use it. The perception teams had on problems they have was different from the one coaches had about them. For example, it was unclear to the teams how learning TDD (whose value is appreciated in the longer term) would solve their *short-term* current and short-term future challenges, such as designing the application for modularity (different sensor characteristics during application-life-time) or security and safety goals provability (e.g., no hard turn while driving fast). Coaches missed the opportunity to help teams solve their problems and use them as a vehicle to improve the teams' technical excellence.

Those who used technical coaching improved their software craftsmanship skills. Unfortunately, the positive effects of improved software craftsmanship skills have a long delay until they become visible in a huge codebase. Considering the LeSS organization's too-fast expansion, the technical coaching had little impact at that time.

Conclusion

The product development system's parts are interwoven and highly influential on each other. It's a system! ADD failed to understand that.

This section will examine some of the symptoms and their influences. In addition to what we present below, we wonder if there were deeper forces at play that we haven't been able to see. Common deeper forces of organizational dysfunction include personal relationship favoritism, sabotaging changes that challenge the status quo of manager positions (so that reverting to the status quo becomes an option), and so much more.

One crucial observation at ADD is that many (described so far) issues were correctly foreseen. Still, many chosen "solutions" avoided (in reality, just postponed) the problems instead of confronting directly and solving them.

One cause of this effect was minimal systems thinking. Mainly due to—often self-inflicted—time pressure, rapid jumping from problem to solution space, being convinced that the causes of problems at hand were obvious. Consequently, leading to more problems, time pressure, and less systems thinking—a reinforcing loop.

Another cause was the many sunshine managers—a delayed effect of lacking a technical career path—who did not

understand and therefore did not warn about or explain any consequence when people disregarded agreements and guidelines or bluntly ignored agreed-upon tasks. Active sabotaging had little to no consequences (for saboteurs), resulting in more dysfunctions. One result was a too-fast expansion of the LeSS organization—an overarching symptom and cause of many other harmful effects.

The retrospective on the timeline view results in five symptomatic categories.

1. Unmet prerequisites for scaling adaptive product development
2. Too-fast expansion to many Requirement Areas
3. Missing understanding of and motivation for Go See
4. Missing motivation for teamwork
5. Missing focus on technical enabling and excellence

Unmet Prerequisites for Scaling Adaptive Product Development

In the beginning, the organization feared that a profound change would disturb the delivery capability of SoP 2018. Combined with *treating products as projects*, it increased the distance between the people involved in SoP 2018 and SoP 2021. The project structure forced the delivery of SoP 2018. Therefore, we started the LeSS adoption, although most people experienced in production-ready car development were unavailable. The technologies, design approach, complexity, organizational design, and ways of working between SoP 2018 and SoP 2021 was utterly different. Therefore, it is very uncertain that involving SoP 2018 people into the first Requirement Area would make the LeSS adoption less painful. However, their experience from really shipping, and the required focus on whole-product overview when a group is getting close to delivery, would have been invaluable in the SoP 2021 non-LeSS organization, especially at the early stage.

The forced "volunteers" led to dysfunctions in the first Requirement Area. The creation of a positive showcase for further scaling was not possible. However, due to running continuous improvement as a *project* (thus, with an *end*) instead of *continuously*, ADD still executed the "scale-up" plan.

More significantly, the importance of a rock-solid build system and continuous integration as behavior was underestimated and ignored because of inexperience in large-scale agile software development and weak software craftsmanship.

The modular-system contract between ADD and the Product Management department and the "we need to deliver everything" ethos led to no prioritization and lower acceptance of reducing-the-scope discussions. The failed PO/APO coaching resulted in a so-called Product Backlog consisting of technical tasks.

Both resulted in a lack of whole-product focus and prioritization, leading to high busyness and *output* of completed technical tasks, but very low output of completed customer features, and thus no useful *outcome*. This typhoon of technical tasks made it impossible for the PO to have a meaningful whole-product overview, to order the so-called "Product Backlog," which disempowered him and made him dependent on the technically-involved "APOs"—he had to believe what the "APOs" told him.

Further, managers and a vast number of junior Scrum Masters failed to establish effective decision-making protocols in communities of practice and teams.

Missing Understanding of and Motivation for Go See

First, the Go See practice was misunderstood or not understood at all. Go See in software development means observing the *code* and the people's experiences in the moment while hands-on creating the code and grasping the situation, positive and negative, from that.

It isn't "Go See in the building where product developers work." It is, for example, sitting in the back of the room while a

team is doing mob programming and grasping the situation by observing the developers in that context and the code they are creating.

Key Point: The heart of "gemba" in software development is… the *source code*.

See also the respective guides, *Guide: Go See* [3, p. 125], and *Guide: Managers as Teachers and Learners* [3, p. 128].

Second, the area of concern for managers was too broad; it defused their efforts and limited their motivation for practicing Go See. Individual, incentivized targets and the individual performance appraisal (IPA) for managers supported this behavior; it did not include teams' feedback. Instead, it was done solely by their line manager.

Missing Motivation for Teamwork

One cause for the missing motivation for teamwork was the unchanged and opaque IPA, in some cases combined with specific persons' bonuses attached to specific goals or achievements. It promoted individualism instead of teamwork on any level.

Missing focus on technical enabling and excellence

The attention gap for technical enabling and excellence led to negative reinforcing loops. The deployment to the actual target platform, or at least an emulated environment, was undervalued mainly due to decades of early software component development and late integration. Instead, ADD used a (not suitable for production) rapid-prototyping platform for a long time. Consequently, the software evolved towards only being suitable for the rapid-prototyping platform, hiding design flaws and undermining the *whole product focus*.

What Should Have Happened To Keep the Magnitude of the Chaos Phase Bounded?

First and foremost, ensure that *continuous improvement* was understood and established as such—no fixed end of "completing the change!" *Continuous improvement towards perfection* is a *life long* undertaking. At ADD, this part might have been rationally understood. However, since *culture follows structure*, peoples' behavior showed the opposite.

Second, set up a stable parallel organization. Have one Product Backlog for both organizations. It would motivate one code base and prevent separation in code. Further, it would influence the entire product group to work on the highest customer value.

Build up the LeSS organization with real volunteers, expand it slowly and carefully, and only when the preconditions are in place, including a prioritized common Product Backlog for all teams and Requirement Areas, and a working build system which ensures fast feedback.

Crucially, follow the standard advice in LeSS Huge to incrementally expand one Requirement Area at a time.

Remove complexity by treating products as what they are—products. Treating them still as projects leads to organizing people around just tasks and creates further dysfunctions. This approach would have brought the people from SoP 2018 and SoP 2021 closer together. People would have been able to benefit from each other's experiences. It would also enable one common Product Backlog for the entire product group.

To keep the pace and magnitude of change bound to a manageable level, starting with SoP 2018 would have been a wiser choice. Within SoP 2018, the number of knowns—for example, target platform, processes, build pipeline—was much higher. Those circumstances would have made it easier to move from single-function and component teams towards feature teams. Further, the SoP 2018 setup change would be confined to one Requirement Area, focusing on the transition towards feature teams, avoiding premature expansion of the LeSS organization.

The individual performance appraisal should have been removed or changed to minimize its negative impact and to support teamwork and self-management. An in-depth education, including Scrum basics, would have laid a more solid

ground for the adoption.

Summary and Positive Effects

With the start of the LeSS adoption, the amount of euphoria was high, and so too were the expectations on the whole new organization. Time passed, it became visible that expectations were far from being met, and the euphoria turned into pain. Where did this pain come from?

> ...in a larger product group (say, 500 people) adopting large-scale Scrum, systemic weaknesses are exposed in the organizational design—in structure, processes, rewards, people, and tasks. [1, p. 290]

Indeed, so did the weaknesses of the prior organizational design at ADD, which led to this situation. It is naive to expect that after decades of building up organizational and technical debt, suddenly overnight, the behavior of a rock-solid build system and continuous integration would be in place. Paying off those debts will take years, and the practices and technical agility will improve gradually.

Technical agility constrains organizational agility—or phrased in reverse "Organizational Agility is constrained by Technical Agility." Therefore, the highly flexible organizational structure there was at ADD, provided only limited benefits. Instead, the gap between how ADD wanted to work and how the technical infrastructure and product code base allowed it to work was the pain. Some changes had to be reverted, and some adapted to get the organization going again (see Figure 22).

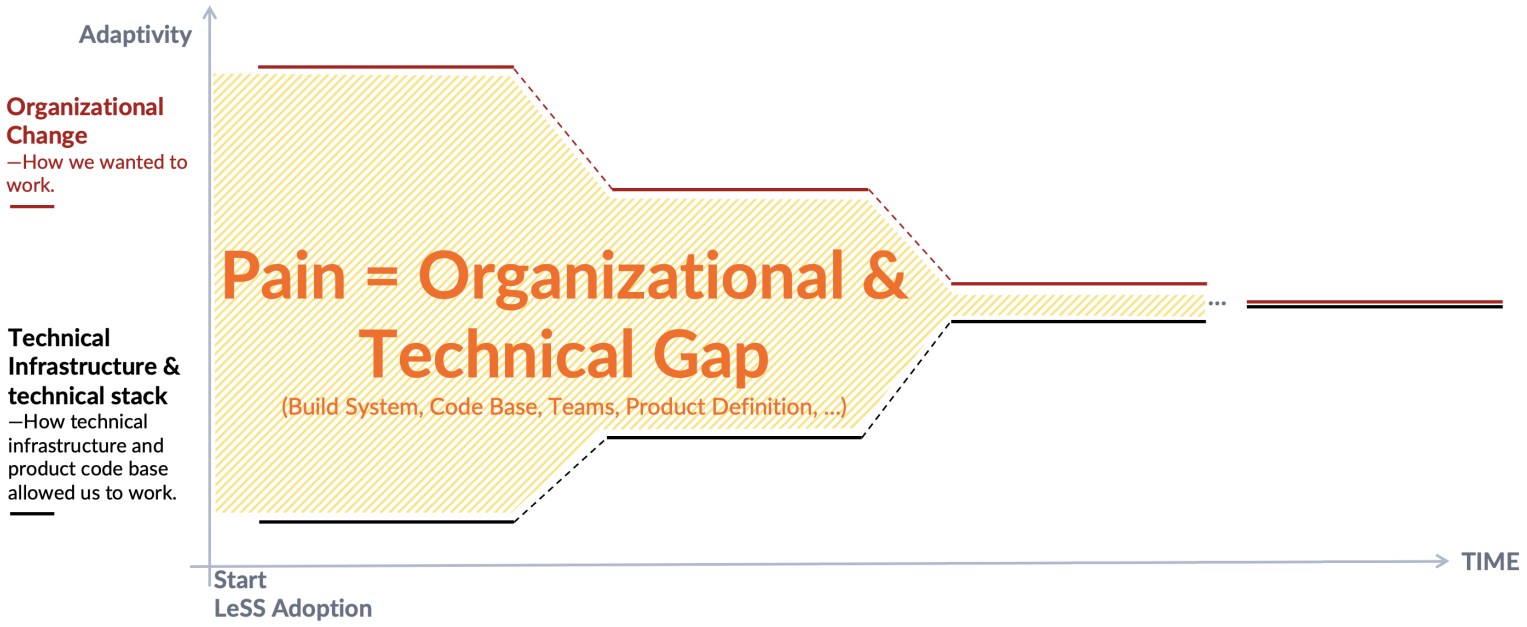


Figure 22: Technical agility constrains organizational agility.

> **Do not expect this to go fast; it will take years or perhaps decades**—in fact, forever, considering the pillar of continuous improvement. A good sense of humor, an informal supportive community of practice, and patience is especially helpful in organizational improvement. Celebrate small steps forward. Especially in the work of organizational redesign, we encourage our clients to **keep in mind systems thinking and the dynamic of local optimization**. [1, p. 284, emphasis added]

Learnings and Positive Effects

A key purpose of LeSS is to make already-existing issues visible again. Hence, when the euphoria settled and the product focus increased, the issues and failures described in this report became visible. Then, to correct the errors and make up for what ADD lost so far, the introduction of inevitable changes began.

Despite the dysfunctions and pain ADD was in, these have led to positive effects and learnings.

An important learning is that *whole product focus* and an *inspiring product vision* are crucial for successful adaptive development.

Collaboration Between Departments

Established cross-area events, such as reviews, Product Backlog refinements, and retrospectives, led to better collaboration between departments. Further, fewer roles and a higher degree of cross-functional teams reduced hand-overs and politics between departments—less "we vs. them" thinking.

Multiple departments, even those adjacent to ADD, are working based on the same principles. Therefore, there are fewer escalations at the department borders necessary.

Higher Focus on Self-Managing Teams and Frequent Retrospectives

Since the LeSS introduction, there is a sharper focus on *teams* rather than individuals, while demanding and supporting self-management. By far, more decisions are made by teams today compared to earlier, leading to less hand-off and information scatter.

Further, retrospectives, inspections, and adaptations occur each Sprint, which hardly ever happened before the LeSS adoption. Meaning root-causes are being analyzed and fixed much more frequently.

Technical Excellence

The attention to technical excellence increased significantly with the start of the LeSS adoption. It led to a higher focus on virtualization and automation.

Coaching on technical excellence increased the awareness of its importance and implications. It led to modern C++, clean(er) code, TDD, etc. Merging the research with the development group amplified the progress on technical excellence. Further, cross-team collaboration improved shared code ownership.

There is also more try, inspect, and adapt to learn from the real product environment instead of finding a "solution" to a problem without writing any code.

ADD Learned to Drive the LeSS Adoption Themselves

When an external coach is engaged to help a company adopt LeSS, the coach usually takes the driving person's role. On

the positive side, this approach leads to fewer failures in the beginning. On the contrary, as soon as the coach leaves, the company must learn to continue alone, which could be hard without a coach.

In this case, BMW Groups' internal people took the driving person's role. The coach focused very much on enabling the internal employees to make the adoption themselves. Doing so led to some failures due to still limited internal experience, which the ADD learned from. This way, the product group can still pursue the LeSS adoption themselves even after all external coaches' engagement ended.

ADD got used to real adaptations. Therefore, it is easier to initiate changes when necessary.

From this point on, the change must be continuous and sustainable, made in small steps, going towards the initial optimizing goals.

Two steps forward, one step back, still brings you ahead!

Outlook

During the writing period of this report, further adaptations took place at ADD. Teams of a Requirement Area and their dedicated line manager are one organizational unit now. Line managers have a much narrower scope of concern and can focus on teams of one Requirement Area. APOs and Scrum Masters remain in other separate organizational units.

Prioritization is starting to happen. The product focus is increasing fast, allowing everyone to inspect and adapt based on product-related issues.

It seems that ADD is entering the fourth phase of the Satir change model—the integration phase. Overlaying the Satir change model pattern with Shu-Ha-Ri stages gives a better understanding of why it is happening this way, and how the future could look like (see Figure 23).

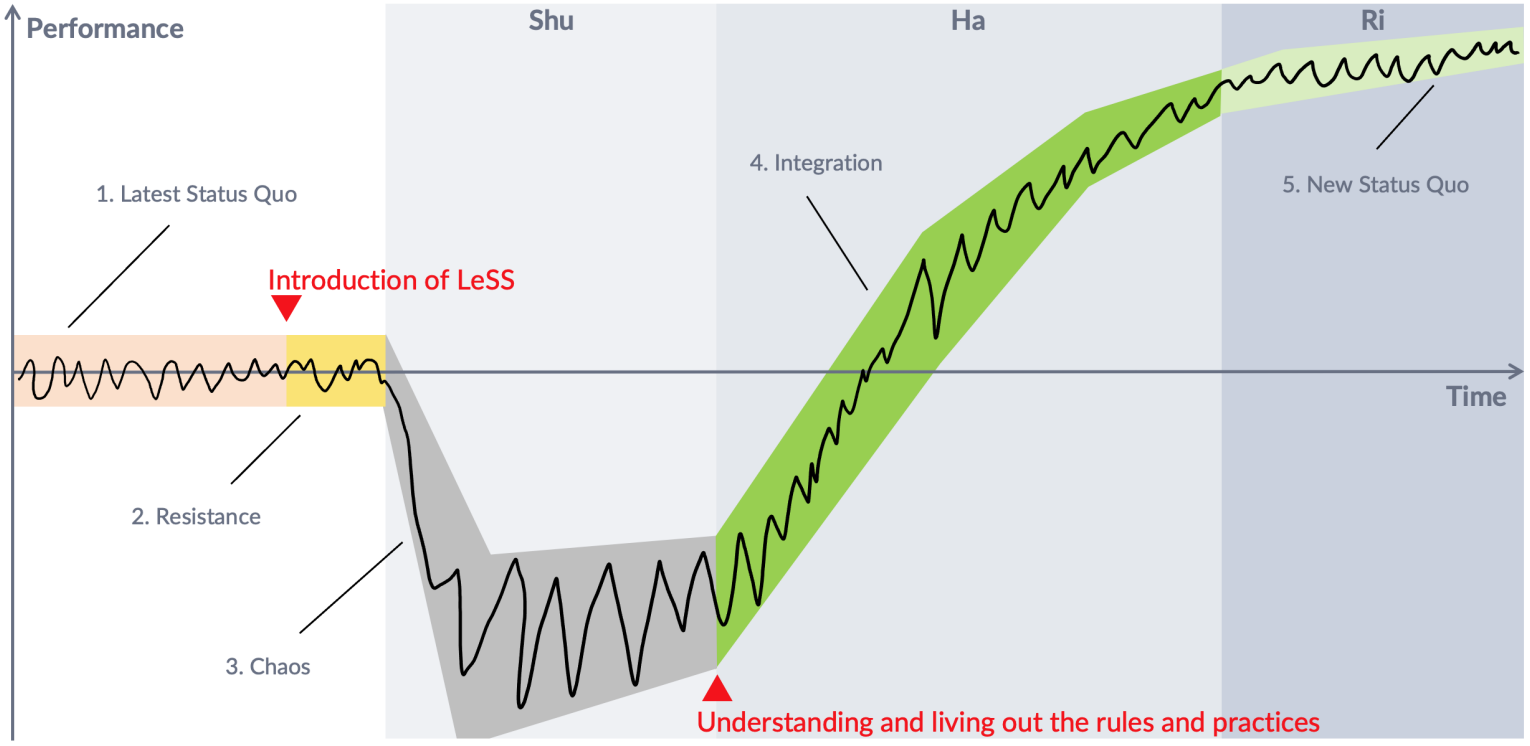


Figure 23: Journey of BMW Group's LeSS adoption overlaid with Shu-Ha-Ri stages. Based on the Satir change model.

Facing the blunt reality, and understanding and living out the rules and practices help enter the *integration phase*, which also correlates with the *Ha* stage. In the Ha stage, people in an organization must reflect on the meaning and purpose of everything that they learned. Therefore, they come to a deeper understanding of the art—in this case, large-scale adaptive development—than pure repetitive practice can allow.

It is too early to draw any conclusions from the recent adaptations. They need close observation and analysis over time. The results of this could be the next volume of this report.

Technology View

In this part, several technical aspects are explored: what went well, what fell short, and what to learn for the next time. To create a better understanding of the starting situation, here's a brief outline of the technical situation before the adoption started. The following chapters provide more details as needed.

Before Adoption

The BMW Group started exploring ADAS decades before the adoption. The exploration routes diverged in fairly disconnected research projects such as Ph.D. studies, outsourcing, and other academic research projects. Most of the time, each of these routes were connected by project plans and feature descriptions, but vastly disconnected in actual design and code. This led to a wide variety of code bases and tools.

By the way, in my role as a technical coach, I (Michael Mai) observe that this is a common pattern when software development is treated as multiple projects rather than as one product.

The academic projects frequently used Matlab and Simulink. The outsourcers that had been engaged by the BMW Group used Matlab or C. This complicated collaboration and also explained some of the divergent concepts in the code base.

Noteworthy with respect to achieving the difficult goal of at least a Level 3 AD, there was no focus on modern machine-learning ("ML", "AI") approaches, in stark contrast to the extremely ML-centric approaches taken by the most accomplished leaders in the field (e.g., Waymo, Tesla). Rather, the BMW Group continued to explore and apply traditional control methods, such as imperative programming with formulas, as had been used previously in relatively simple driving-control problems, used in cruise-control.

The BMW Group centralized some of their research and development effort into three bigger in-house groups: SoP 2018, SoP 2021 highly-automated driving (HAD), and SoP 2021 fully-automated driving (FAD). Those groups did have different knowledge, experience and exposure to vehicle-grade product development. Also, those groups joined at different times into the adoption (see Figure 24). See also Figure 17 and Fake Volunteers.

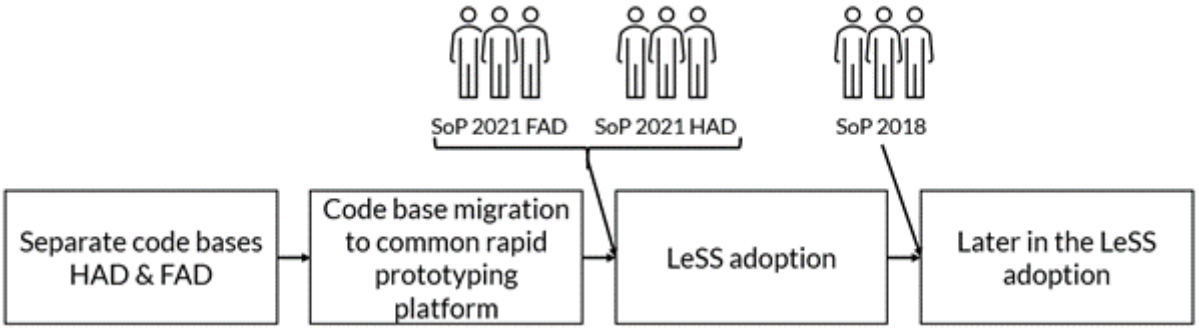


Figure 24: Joining of groups.

The SoP 2021 groups, which consisted mostly of researchers and new hires from university, collaborated closer and formed the starting seed for the LeSS adoption. So, *there were few experienced software developers, with limited experience in building products for vehicles, and with compartmentalized domain knowledge in ADAS*.

In preparation for the major ADD initiative, management assumed that growing the group by new hires from university would introduce modern software development skills, and modern clean and well-designed code, into the code base—another indicator of managers lacking proven skills in software development. Given the limited experience that those new non-top-tier joiners (from university or from the market) could contribute, it was a hope in vain. Chapter People explores additional reasons.

The inexperienced SoP 2021 groups made the fundamental architectural decision to directly couple to the robot-research-based message-oriented middleware platform ROS. ROS is used for many research papers on robotics, but the design philosophy of ROS is incompatible with the vehicle target platform.

Because of the (1) bifurcation into HAD and FAD project groups, and because (2) they were organized into multiple component teams (the traditional model that the managers knew, from building physical components), and because of (3) the lack of skill in software development and large-scale coordination, there had been *essentially no alignment on software concepts and design across the multiple code bases these groups created. And those groups' divergent code bases only interacted at the level of ROS messages*.

Due to differences in project objectives the "alignment" in concepts and software was rated lower priority—despite its obvious essential nature for delivering a real product. Yet another sign of the lack of experience amongst management and developers in large-scale and complex software development. So, both the HAD and FAD code bases were very divergent. In addition to this difference, both groups independently organized themselves into *component teams*, as previously promoted by BMW managers for their prior relatively trivial and smaller software initiatives, naively unaware of the risks and lack of learning and adaptiveness this created when trying to create a difficult research-intensive end-to-end AD robot that in reality would need lots of learning-based feedback and change. These reasons led to little alignment.

The SoP 2018 group was the veteran *traditional* engineering group with product development experience (see Unmet Prerequisites for Scaling Adaptive Product Development for why this group exists and why the LeSS adoption didn't include this group from the start). Due to their deployment-target-constrained toolchain, they had to use a restricted subset of C++, and were largely a group of traditional C programmers. They had little or no experience in modern and large-scale software-intensive design approaches based on more recent industry standards, such as object-oriented design (OOD). This lack of OOD skill contributed to the decision not evolving the SoP 2018 code base into the new product's code base. Note that without that skill the developers could not effectively design modern C++ code within a new system orders of magnitude more complex and different from anything they had done before.

The different sub-groups struggled on shared software design and concepts in their communication and thinking. Their

distinct past, goals and code base did not prepare the groups for joint development. Still, the objective of management was joining these groups, despite their differences, into one. Why? For (1) these are BMW Group employees and should be enabled to work on a new product, (2) it was mistakenly assessed that further development would require a substantial increase in staff (see People for details), (3) harvesting the fruits of decade long research and putting it into a product—which requires more than just the researchers, and (4) closing the gap between research, gaining insights, and putting the insights into a product for customers.

And picking up on point (2), "big staff for big work", most of the BMW management had a background in mechanical and electrical engineering and production, and incorrectly viewed software R&D like a *manufacturing* problem rather than an R&D problem, where *in manufacturing more bodies can mean more progress*. But of course, *in software R&D, adding lots of bodies rather than the right talent, aggravates problems and progress*. So, yet another sign of lack of modern and large-scale complex software management expertise, starting at the board level.

The following sections focus on organizational, political, and personal dysfunctions and system dynamics that resulted in this challenging technical landscape. Some ideas and tried proposals to overcome this situation are included and discussed as well.

One Product, One Repository, One Branch

Before the LeSS adoption, there were more than 150 repositories and several hundred branches. As mentioned, the design and repository structure of the code base was strongly influenced by the fact that the groups worked in multiple projects, and as component teams.

Another sign of the lack of experience in modern large-scale adaptive development, by both management and developers, was the lack of awareness of the many problems that would be caused by all those repositories and branches when the disparate groups had to come together and ship one common product. The old multiple-projects and component teams model delayed seeing those problems by delaying integration.

Fortunately, one of the first steps of the new ADD department was the definition of the product. This definition of a single product started to unify those groups and create the motivation for one repository, one branch. The main requirements that led to the decision of one repository, one branch were:

- Focus work effort for the product
    - Otherwise milestones are endangered
- Consistent implementation of legal requirements and amendments
    - Otherwise the product is not legal to sell
- Consistent implementation of business and user experience requirement
    - Otherwise the product is unable to sell

The chosen versioning tool was GIT, where "one branch" means "master". The practice to work only on master (the equivalent is *trunk* in many tools) is called trunk-based development.

However, there was still a major delayed-integration problem: Previously, the groups focused *separately* on Level 3 and Level 4 driving. For commercial risk-management the Level 3 functionality needed to be stable first, since the vision was to at least offer a Level 3 AD, and therefore it had a higher management attention and influence on Product Backlog ordering. Still, many managers realized that if Level 4 development would have been dropped, the race for Level 4 in the market would have been lost to the competitors. So, *both streams were followed in parallel*. I will explore in appropriate sections these opposing mindsets and their harmful impact on integration, and more broadly on product development.

Internal Partners and Deviation From the One Branch Rule

The energy generated from the single product definition for ADD to merge repositories and branches also drove the separation from other departments within BMW Group.

The separation revealed the desire of managers to differentiate one from each other. In the perspective of the other managers, the ADD manager created a super department that consumes other departments—meaning also consuming their own department. Department reduction is usually in the interest of the product and higher management, but rarely in the interest of (middle) management heading these departments. Therefore, the corollary of one/wider product definition is the need of other department's managers to distinguish themselves, for example by *local optimization* in their department to justify continued existence.

For example, the department tasked with human-machine interface (HMI: displays, head-up display, steering wheel, radio control, etc.) development argued that it also had other vehicle projects as clients. Therefore, their management argued (1) centralizing design capacity is "optimal" (of course, actually a *local* optimization), (2) splitting the design team and joining the ADD group would disrupt their multiple HMI *projects*, and (3) if the design team would have been dispersed then every project's timeline would be in jeopardy. Needless to say, this traditional HMI and design department is—like many other departments in BMW Group—organized to control *suppliers* of design, user-experience studies, and related software and hardware.

The way BMW Group organizes vehicle development, vehicle manufacturing, and vehicle projects is also used by departments to claim independence from ADD's product vision. The unspoken argument is that the existing status quo (local) optimization shouldn't be changed or challenged by any "super" department.

Since the organization design of an overall system's inter-department collaboration is the responsibility of C-level management, and this system was apparently not understood by them to be problematic (and thus not improved) for the success of ADD, it illustrates that C-level management also did not have the knowledge or skills to manage the internal creation of very large and complex software. It is probable they continued to view the problem as requiring their traditional supplier-management model.

Research Collaborators and Deviation From the One Branch Rule

Integrating research results into the product and the product code is important for ADD, therefore ADD maintains several such research projects.

Management decided for separate repositories and therefore long-delayed integration. Some reasons why management decided for this were (1) high degree of uncertainty which proposal under research would finally go into the product, (2) several non-BMW Group researchers were involved; therefore, management limited the amount of unrelated BMW Group's code, (3) unacceptably poor software design and code quality in research code, and (4) researchers experimented with different programming languages.

Some of the consequences of this delayed integration were (1) divergent concepts, (2) implication for subsequent algorithms, data structure and evaluation, and (3) vastly different understanding of the problem and details the research code solves, which led to challenges during integration and adjustments of the product's software design.

External Partners and Deviation From the One Branch Rule

At BMW Group, a common approach is to slice the code base into pieces (one slice/piece = one repository) for each partner (depending on the contractual collaboration level). ADD management adopted this naive approach. The issue with this is (1) it emphasizes conforming to the existing (traditional) contracts model, (2) therefore it is locally optimized for contracts, not product success or end-to-end collaboration, (3) by optimizing for the traditional partner-contract

model effective integration and collaboration is impaired, and (4) the total solution, which requires adaptive collaboration and frequent integration for success, is severely hampered. All this is a sign of missing experience of modern large-scale adaptive development practice, lack of understanding of how much more complex this initiative is, the kinds of collaboration and integration and repository practices needed, and senior management more concerned with “good” contracts than working software.

“Off Car” Functions and Deviation From the One Branch Rule

Even though there was now a “one product” definition, due to the lack of insight and experience by management and developers, they continued with delayed integration and multiple repositories or branches for a variety of components and “projects”.

For example, the data center and data retrieval related components hadn’t been integrated with the regular product development for long periods. This affected the data retrieval and reprocessing capability of the data center and processing cluster. Due to this non-integration, mass-processing of data for validation or simulation was not possible due to incompatible data or not useful data due to obsolete (because of the delayed integration) code used in cluster processing.

Only after the issue was painfully obvious enough, did developers and managers start merging efforts of code and people. Yet moving people around—another quick fix—to deal with these self-inflicted wounds discouraged management from thinking about the proper organization of people and underlying causes, and therefore the staff size of ADD increased even more.

Platform and Vendor-Specific Code and Deviation From the One Branch Rule

Senior management had a vision of ADD being a platform that could be used by OEMs. Given the high-level R&D difficulty of the problem and the unprecedented experience and staff at BMW, this was arguably wishful thinking amongst the deal makers and C-level.

In any event, wishful thinking won, and management decided to split platform-dependent (=vendor specific) and platform-independent code into separate repositories, since they and the developers did not understand or have the software skill to separate the pieces architecturally while keeping just one repository. The split resulted in even more complicated development and delayed integration issues on all participating OEM parties—including ADD.

A Rushed Merging of Repositories

The first Requirement Area started with a subset of the 2021-release code base; it also consisted only of the first batch of Fake Volunteers. The ramping up was decided by management even before the beginning of the adoption—also referred to as organizational cut-over. Management explicitly stated that things that worked before, should also work after any employee completed joining the adoption, otherwise those employees would be more productive and kept in the old structure. This led to a very hastily merging of multiple repositories (code bases) that had little alignment. Other than pushing to rush it, management left the precise procedure to the developers on how to achieve this.

Due to this management directive and their push to “go faster”, there was a rushed big-bang of putting all repositories at once into one repository and figuring out rectification and alignment later. That approach was based on reproducing steps from the old build system in stitching build artefacts together. This stitching-big-bang approach did not create the

time or interest for aligning software design or misaligned concepts and just deferred these critical problems. This would lead to many problems later, and illustrated the classic “faster is slower” dynamic so often pushed by management teams.

Software Design and Architecture

First, some broad context and influences:

As discussed, most developers had been researchers with ADD. This influenced their choice of software design. Originated in an environment that rewards *single*-person-excellence by for example handing out Ph.D. titles, their natural grasp of development is in isolation. This resulted among other things in the many repositories before the big-merge.

Also an influencing factor: being part of a large corporation influences mindset. One is that of clinging on to weak solutions due to a *sunk cost* bias.

Some more context: (1) the code base should be maintainable for 20 and more years, (2) legal obligation influences the update cycle, (3) variability of sensors, (4) variability in protocols and semantic primitives among different OEMs, and (5) ongoing research and experimentation.

Explicit Coding or Machine Learning? Expertise and Hiring Dynamics

The general architectural approach of ADP is *explicitly coded* software. Especially in contrast to other important vendors and their reported successes, failures, and their increasing focus on close to a “100%” ML (“AI”) approach to autonomous driving: why is ADD sticking to the traditional alternative?

The origin of most developers is research. The tendency in their (German) academic experience is working on one hypothesis in detail with a strong emphasis on one-person excellence. This develops perseverance and dedication, but also narrow-mindedness. This leads to *reducing* options for solving the challenge of autonomous driving. Some of the options are ML or a combination of explicit code and ML.

Why did the management not question the current approach more rigorously, especially since the global leaders were increasingly publicly emphasizing that a key lesson learned was to focus on end-to-end ML? Firstly, most managers were blinded or biased by their own (German) academic career.

Secondly, they had unwarranted trust in their researchers. Only a few managers have experience with large, complex and adaptive software development, only a few have the necessary background and skills to acquire this experience, and most *are reluctant to actually visit the place of actual work and look into the code and grasp what is really going on in the details of the software* (see Missing Understanding of and Motivation for Go See). This combination leads to the belief that the employees, especially those with *academic* degrees, are experts in actually creating large and complex software, and skillful design for AD. Due to the lack and reluctance to question their “experts”, they just went along. There was no serious questioning of the status quo, the “experts”, or investigation of what external experienced experts were doing.

Thirdly, some managers did have suspicions if the chosen path was the correct one, but didn’t manage to challenge the internal authorities, or their manager-peers, to seriously explore alternatives. The reasons for not following up their suspicions are manyfold, cannot be discussed here in full, and would therefore mostly be speculation, but it is a common pattern in business that managers do not raise alarms that challenge the status quo, to avoid disputes or harming their future career prospects.

Fourthly, consider if the managers would have decided to actually favor another approach, based on ML. Which employees would have been able to carry out the implementation of the product, and what would have been the impact on the timeline? None had the skills and experience, and “it would take too long.” One of the early technical coaches,

Craig Larman, on recognizing the lack of talent, had early-on recommended to the senior management a risk-mitigation strategy to start a second parallel initiative staffed with the very best global experts that money could buy, including ML experts, but this was rejected.

Given that many of the developers had been researchers from an academic environment honoring the hero-paradigm, with overwhelmingly explicitly-coded solutions in their theses, turning down alternatives is easier than committing effort to explore the dramatically different approach of an ML-based architecture.

Why was turning down alternatives easy for developers? Researchers are recognized by their expertise. Building sufficient expertise to question a familiar path takes time, opportunities, and dedication… items not easily found in a competitive peer-to-peer environment in a company, especially with command-and-control structure, and personalized bonus systems which don't honor exploring alternatives. Permitting an alternative outside view is almost equal to confessing of not being an "expert"—something most academics avoid.

Why did ADD only consider their "experts" (researchers) opinions and not the opinions of modern-skilled software developers? Well-established and successful companies with a strong track-record in traditional engineering solutions, consider their own as experts. Asking top-decile externals challenge their self-understanding.

The typical thinking of managers: long-term employees provide solid arguments for long-term products as those have to stick to the decisions long-term. This thinking is misguided as the connection from decisions to employees' stickiness is false. Employees normally have other strong reasons to stay, for example family, friends, local ties, and company fringe benefits. *Externals opt to stay*—if permitted by budget and managers and are therefore a better indicator of wise strategies than internals.

Why hadn't ADD hired top-decile software developers? Top developers tend to favor (1) self-managed work (very limited manager interference), (2) working closely with other top developers, (3) working for purpose, (4) working for appreciation, as well as (5) be unwilling to work for traditional and software-unskilled managers.

Putting a light on the hiring process with regard to software *design* approaches reveals an unintended effect. BMW Group has well established processes for staffing and hiring. By *BMW Group's C-level management decision* to operate their AD development as a department with close ties, ADD inherited those hiring processes.

Essential components in the hiring process include interviews by potential future colleagues, and a programming exercise. The selection of the interview partners and the programming exercise plays an important role in how diversity in software engineering and software design approaches is fostered or crippled. Imagine, a candidate is interviewed by a BMW staff member with a strong traditional, non-OOD and non-ML mindset. Any candidate with strong OOD or ML skills has *contradictory* design advice. Also, if the programming exercise is drafted to test basic programming skills and no challenges in large-scale design or applying OOD or ML, the candidate gets the impression that ADD poses no adequate challenge and is no place to prosper.

Now, ADD does have a few researchers with some academic focus on ML and supporting infrastructure. They are also assembled in an ML Community. But to make a difference they need to have substantial proof (driven from data). Generating strong "proof" requires a massive amount of time and capacity—yet neither is provided by skeptical managers.

Finally, the whole discussion on explicit vs. ML algorithm is overshadowed in Germany by the unclear legal situation. It is not clear if ML is allowed to make decisions for driving a vehicle. In the current situation, the BMW Group manager's approach is to conform to traditional approaches in fear of unclear legal repercussions.

Modular Software and Integration—Deliberate vs. Coincidental Design

The many separate code bases before One Product, One Repository, One Branch complicated the creation of collaborative components and delayed integration. The resulting integration nightmare was worked around with many tweaks, fixes, and temporary actions. Most of these "fixes" were abandoned as soon as the integration phase was

completed—most developers ignored those “fixes” as they didn’t contribute to their component’s core capability.

One of the root causes for the integration challenges were divergent concepts, ideas, and software design, which in turn were a result of the system in place before the adoption started. None of these were tackled during the rushed merge of repositories (see A Rushed Merging of Repositories).

Why didn’t the *managers* address the design issues? Their primary focus has been getting all developers into direct disciplinary (and functional) control again, which implied a focus on connecting disconnected and disparate code, not on harmonizing design and concepts. Also, those managers had overconfident beliefs on easily solving these hard issues, due to limited modern large-scale software development experience.

Why didn’t the *developers* address the design issues? The guiding principle for most developers at ADD is their component, since as discussed they mostly organized as component teams and had no experience working together to create a cohesive large-scale product.

Not Asking the Right Questions

BMW Groups used to outsource most of its software development. ADD is the first modern and really large-scale adaptive development at BMW. Therefore, *ADD’s experience and ability asking guiding questions which lead to a good long-term software design is limited*.

Why is asking good guiding questions required for designing a good software product? Questions provoke and frame current and future use of the product, current and future flexibility, current and future challenges. Unskilled questions diverge time and effort. Separating good from unskilled or not helpful questions is typically only mastered with experience, domain-insight, diversity, sufficient foresight, and grasp of the market.

Which systems dynamic led ADD to not asking good guiding questions? The strong emphasis of *outsourcing software-related work* contributed to this inability. BMW Group tries to understand the problem space, dissect the problems into subproblems, and hand those to vendors. The issues here are: (1) by slicing the problem space into smaller chunks the risk of losing track of the original problem is huge (especially when a skillful overall solution is not actually known for autonomous driving), (2) opportunities from “lower” domains to solve the original problem won’t be incorporated into the original problem, (3) the separation leads to suboptimal solutions, and (4) the separation leads to narrow-minded thinking.

Green Build

A “green build” is a combination of (1) completed integration, (2) completed automated code quality checks without any faults, and (3) all completion of all prescribed tests (unittest, integration tests, integrated tests, (sub)system tests) without detecting any faults. The “green build” indicator is therefore an expression of the organization’s ability to produce a product continuously and therefore its capability of generating strong and salient feedback for learning and adapting, and… potential revenue.

Build System Before the Change

Before the beginning of the adoption, the groups were organized for independent work (Figure 24) which was reflected in their code bases as well as their build systems. They simply stitched build artefacts together which only occasionally worked due to integration issues.

Key point: Not every component-build or piecewise-build leads to agile end-to-end development discipline with short

feedback.

With the rushed merging of code repositories (Merging Repositories) the build system had to be subsequently changed. Revealing inconsistencies in build processes, tools, and infrastructure.

Evolving the Build System

Another sign of the lack of skill was the difficulty of getting a new effective build system in place. It took several external experts and veterans of software-based product development to get across that “never change a running system”, which is deeply ingrained in some BMW Group’s managers, is one of the surest (if not fastest) way to *get stuck* also applies to build systems.

The crucial part of a good build system: it provides fast feedback on the integration of the product. Even if multi-stage build techniques are applied.

Why did the PO, APO, and management stick to the old insufficient build system? Partially due to the habit of management at BMW Group that views everything as a project. The tendency of thinking projects implies thinking in terms of: a start, an end, and nothing in between. This thinking discourages frequent feedback, and therefore encourages thinking of a fast integrated feedback as waste.

Why hadn’t there been any movement from the developers to improve the build system? Here are several aspects at play: (1) The personal incentivization plan (IPA) didn’t favor such activities unless a manager had been convinced prior, (2) the work of the developers was optimized for working in isolation, (3) the developers had been tasked with details rather than a product, (4) they hadn’t been developers for the most parts, but researchers with focus on their research topic, and (5) there had been very few developers that had experienced the pleasure of fast integrated feedback from a good build system. In addition to that (6) exchange with other groups was mostly on research topics and not on product development topics.

Why hadn’t there been more guiding conditions to align on integrated product development by management? Foremost, management at BMW Group thinks in terms of departments, not in products. Even vehicle projects are thought of (1) projects and (2) which department is working on. Only with several LeSS for Executive trainings this started to change at ADD. The vice president of ADD started several months after the start of the adoption, insisting on reviewing the Sprint result in the vehicle—this got the “Build and Integration” Community started and searching for better alternatives to the old build system.

One interfacing department tasked by upper management with integrating code to the vehicle was experimenting with Bazel, so Bazel was the prime candidate to replace the old build system.

Under the Radar

The proof of concept was done under the radar, meaning it hadn’t been on the Product Backlog. Why was it needed to do it this way?

In a healthy Scrum and LeSS environment, the Product Backlog consists of improvements and items to be implemented. The process of getting anything into the Product Backlog at that time was heavily influenced by the previous working model, meaning: project requirements, project managers, and very special non-development individual contributed items.

Also, in a healthy Scrum and LeSS environment, there is no need to discuss technical details with PO, APO, or managers apart from implications regarding features and the product. So, no need for Product Backlog so-called ‘items’ for refactoring or experimenting with technical details.

Why did the teams seek approval from their APO for this kind of technical work? In a research-heavy environment, the need for qualified data to prove a point or starting an approach is high by peers, APOs, and management. Generating qualified data is time consuming, in the case of evaluating an alternative build tool, its potential and weaknesses, it requires forking and migrating a relevant piece of the code base. This is nothing that can be done with one hour a week of playing around, but takes a significant fraction of the team's capacity. When talking about "a significant fraction" the BMW Group's company culture calls for approval from "higher" authority. In a Large-Scale Scrum environment, the (A)PO appears like such a higher authority, even as this (A)PO should not be concerned with technical details.

What is the consequence of having such research and exploration tasks solely on the team's Sprint Backlog without reference to the transparency of the Product Backlog? Normally, small research and refactoring tasks resided only in the Sprint Backlog (rather than the Product Backlog) without any hassle, because they would just create lots of minor noise for the PO or APO in the context of large-scale development, which has so many of these minor tasks every Sprint, forever. On the other hand, as explored in the LeSS guide "Handling Special Items" [3, p. 207], *big* "engineering improvement" or research tasks that involve *major* investment decisions or that *beneficially educate* the PO or APOs, or that lead to *useful conflict* discussions with the teams, are worth putting into the Product Backlog. The difference here is (1) magnitude of change, (2) familiarity with the change, and (3) implication on company processes. Changes in this area are not common and therefore require additional learning from anyone working on this item. Bypassing the Product Backlog shortened the normal learning, validation, and improvement mechanism. The consequence of this "avoid the walking and talking to APO" and putting this task into the Product Backlog, are delays, disappointments, frustrations, and magnified confusion.

Why did the team think it is a good idea to put this experiment under the radar instead of putting it into the Product Backlog? ADD's PO and APO have a heavy focus on incremental features rather than the product-as-a-whole. Is there a difference? *In terms of the product, there is no difference; in terms of a product's feature catalog that had been agreed upon with the upper management, the board, marketing, and other departments, there is a great difference between features and the product*. In other words: the APOs in ADD at that time did a local optimization for outputting features in isolation, but not features in a viable product… that can be viably built with a good build system.

The difference plays out here in a way that at that time, almost no "special items" were present in the Product Backlog. And the process of getting potential items for the Product Backlog was designed only for *feature*-related items. So, the fear of the development team was, if they started the walk to the APO for this particular experimentation item, they would be postponed until after SoP and therefore postponed… for about *two years*. Another classic "faster is slower" dynamic, aggravated by management without an understanding of modern build systems or modern building of large-scale software, when the aim is adaptiveness.

This situation also highlights the entanglement of technical transition and organization, insufficiently utilizing the idea of Communities, and the miss-practice of utilizing teams up-to 100% (see "Avoid…Fake queue reduction by increased multitasking or utilization rates" [1, p. 99]). Later, Scrum Masters with the support of the teams, pushed that Communities contribute Product Backlog items and eventually are heard by PO and APO about their opinion on ordering regarding other Product Backlog items.

"Rogue" Unofficial Group

After presenting the promising results of an experimental build system to APOs individually, interest was raised, but due to an initial estimate of 2-3 Sprints to complete the migration, no Product Backlog item was created and there was no Product Backlog refinement to improve the estimate. To the credit of the APO, they believed their developers; but they didn't confirm the understanding of the developers regarding scope, process-integration, influence on the working environment, migration effort, and central infrastructure—despite the obvious too-small estimation.

As a result, no APO maintained the build-system experiment in focus. As a consequence, there hadn't been any team

dedicated to this experiment. Only some developers from different teams joined and worked on this topic. This time with keep-me-informed by some APOs.

Management Push

The accumulation of shortcomings reached at some time "real" management attention. Management finally acknowledged more decisive support, which led to the creation of three temporary groups with dedicated focus:

- Complete migration from catkin and cmake to Bazel
- Improve build speed
- Stabilize build reliability

Key point: Avoid creating new (temporary) structures; prefere providing clear focus and abilities to inquire support for existing teams.

By assigning these tasks to temporary groups instead of permanent feature teams, management constrained their organization to learn. In a healthy LeSS environment, experts would be invited by the tasked feature team to work on the new topic and learn from the expert to the extent that the feature team can then maintain the system or even improve it on their own. Experts may be external or internal Travelers.

Why did management staff those temporary groups with members from a variety of feature teams? APOs are committed by their IPA to certain feature completion delivery at the end of a release, if complete feature teams would be committed to improve the build issues, the associated APOs would fail on their personal incentivization plan.

Also, the behavior of forming temporary groups reveals the mental pattern of classical project management of "throwing experts at the problem to solve it," which only solves the issue in the short term. LeSS, on the other hand, advises the route of *setting the right conditions for those issues to be fixed long-term*.

It Is Not My Build System

One fascinating dynamic was that most developers did not consider contributing to solve build system issues. The reasons for that lie mostly in the organizational system in place before. Before, there was strong separation of responsibilities and accountabilities. Team A is responsible for road modeling, team B from braking, and so on, and also Team Z for the build scripts; and even a complete department dedicated to build-related issues. This enforced the mindset of "it is not my job".

Therefore, it is remarkable that developers started consenting to their coworker's proposals in that area. *It reflects a change in company culture: accepting peer-generated proposals is essential*.

From a LeSS point of view: this demonstrates the difficulty in taking ownership in a company's culture not yet fully transformed. The adoption of communities as a utility to self-manage on a multi-team level was (and still is) in progress.

Lack of Manager Support for a Green Build Solution

Given that the build is neither a product feature nor a technical detail within the product, who is responsible and why didn't they act? Managers are responsible to create an environment that empowers teams to build the product and take accountability on removal of impediments that surpass the powers of the teams—therefore managers are accountable for the build.

In the past many BMW Group's managers outsourced this responsibility and therefore lost insights and ability to improve the build system. Some ADD's managers cared enough for the build and struggled with scheduling tasks. Why? ADD is new to the concept of self-management of teams, therefore it faced confusions on how to handle such situations. Challenges such as (1) who is responsible for detecting issues with the build system, (2) where to report those issues, (3) who is responsible for proposing and finalizing solutions, and (4) how to get any solution implemented.

The struggle originates in the old organizational systems, its formal structure, and the company culture it had cultivated. Traditionally managers would be assigned, in the new model enthusiasts are to sense and respond on their own. LeSS advocates the structural element of "Communities" to support the self-management on cross-team level. The "Build and Integration" Community was founded a few weeks after the beginning of the adoption. Still, the company culture was slowly changing which resulted in several other consequences, more on this in chapter Communities.

Component Build and Code Structure is Coupled to Builds

Another core aspect of a Green Build is the actual code, its structure, and usage by tests. In an environment that continuously modifies the code base, a strong emphasis rests on its structure, readability hence understandability, and ease to evolve. Following the struggles of the code base's origin, neither the structure, nor the understandability, nor the ease to evolve is a given at ADD.

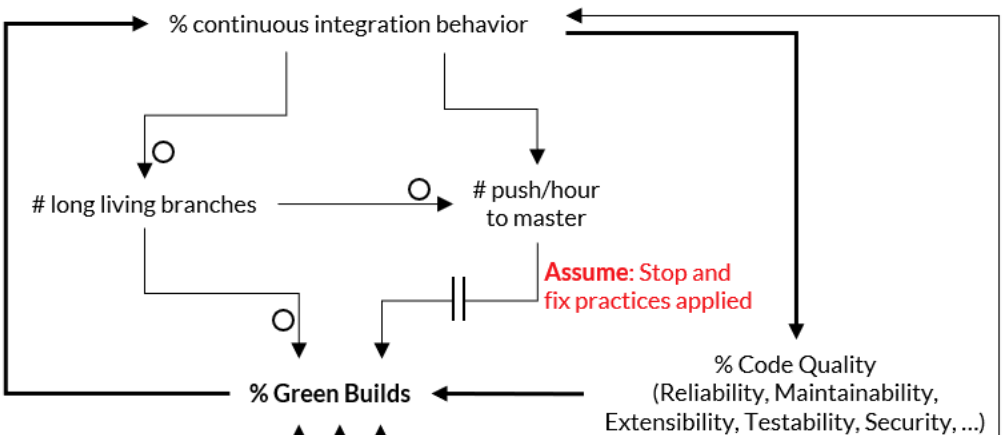
The highly-coupled components created by component teams with weak skill to collaborate and evolve a good overarching design, complicated the creation of true unit test, true component test, and true component builds. The consequence led to long-running and error-prone builds and tests. Several technical approaches to fix this situation had been tried, but failed as they didn't tackle the root cause.

Why was there more focus on components, rather than collaboration or the overall product? The answers vary but are typically (1) classical motivation, and (2) research-like focus and mastery in single topics which materialize as components.

Why hadn't management requested focus on collaborative components and the product? Managers didn't directly prohibit needed deep changes, but the context they provided left little room for deliberation processes, re-designing the code base, and tests. Another classic of "faster is slower" dynamic, aggravated by management without an understanding of modern build systems or modern building of large-scale software, when the aim is adaptiveness.

Would a technical-complexity-control manager have prevented or solved this situation? The question in itself reveals a deep misunderstanding of the system dynamics: Technology (usage of technology for implementing a thought) is only one part of the dynamic; people, organization, and processes are the major parts. Thus, *the framing of the question already diverges from an appropriate solution*. Aside: the question also reveals a confession, that they (1) lack Go See, and (2) lack experience in modern large-scale adaptive development.

Long-Living Branches



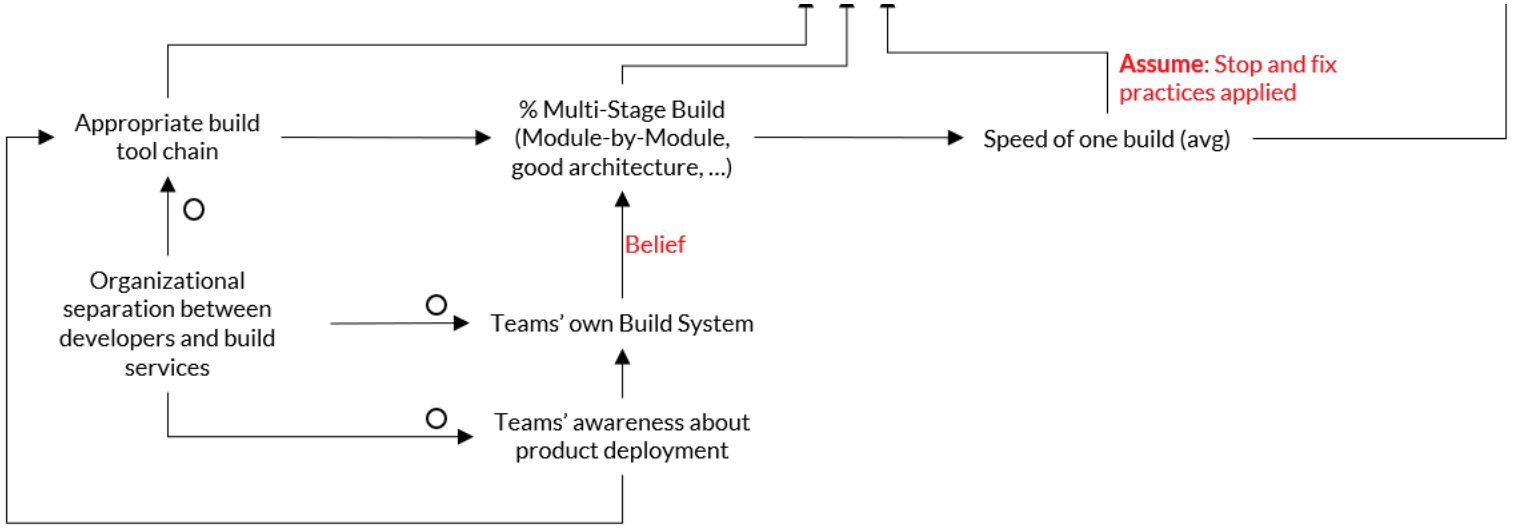


Figure 25: Dynamics of a Green Build is influenced by developers' ability to integrate and test changes fast and frequently.

ADD's developers with their limited experience in large-scale development and limited experience in skillfully designing large-scale software systems struggled working on one branch. Their past working structure fostered working on separate code bases (and repositories). The rushed merge didn't change the fundamental developer's experience so they continued to struggle working on one branch with hundreds of teams.

This inability to collaborate effectively on one branch leads to many long-living branches. Why? First, the old system fostered single-minded excellence in components which resulted in isolated code bases. The closest resemblance with isolated coded bases in a mono-repository are long-living branches. Secondly, most developers are actual researchers and no programmers. This leads to missing skills in design, coding, incremental developing software, and effective communication and collaboration in and via code.

The lack of sufficient modern experienced developers resulted in a systematic inability to self-educate on how to develop incrementally on a large code base with one branch.

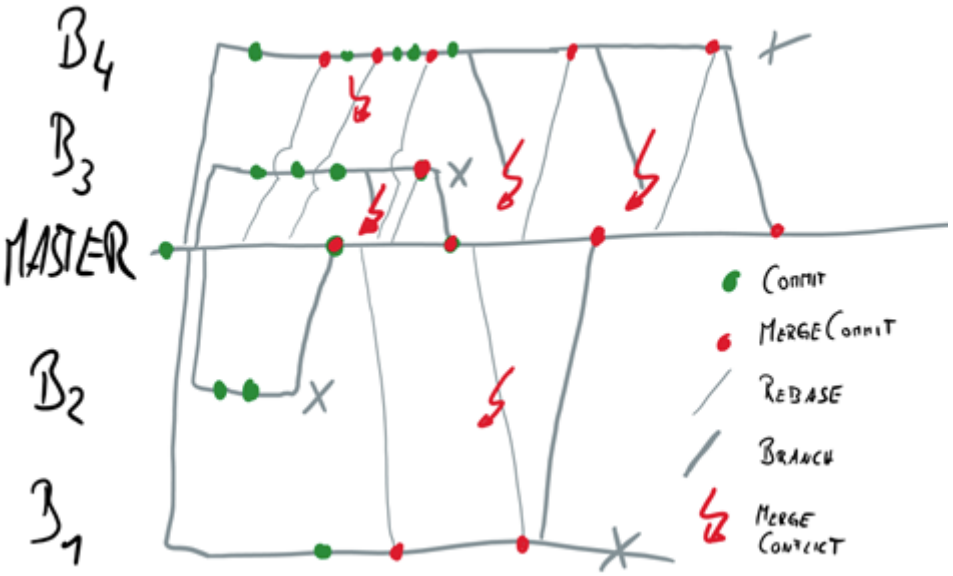


Figure 26: All subsequent merges of long-living branches to master incur a high likelihood of severe merge conflicts.

The resulting huge and complex patches while merging those long-living branches to master (see Figure 26), led to challenges for the already highly-loaded build system. Why? The previously discussed lack of component builds and tests led to whole system clean rebuilds and retests each time. Due to the group's prior focus on components instead of units, many tests incurred running verification tests with time-consuming simulation.

The long running builds were also one reason for long-living branches. Why? Due to cycle-times of 4-8h on avg. from push to red/green, many teams adopted the malpractice of accumulating many commits on their long-living branch and then trying to punch a merge to master-branch once a Sprint. A disaster, of course.

One consequence of those complex big-batch once-a-Sprint merges were frequent red builds. Why? As soon as several long-living branches in a poorly-designed software system are merged, hard-to-avoid/resolve conflicts occur as the inconsistent design of the components is exposed. Each merge conflict requires manual correction and re-merging to master, which triggers the same pattern again, with just one commit less.

Extensive Conformity Checks

The not-owning-the-product mindset of many developers (driven from component teams and management behaviors) led to the system of builds enforcing extensive verification for conformity and testing its viability. Therefore, conformity rules (Figure 27) and good development practice for large development on product level haven't been followed. This resulted in manager's quick-fixing of the system, by introducing mandatory automated build steps for conformity and rule compliance for each build. Why for each build? Traditional management thinking is calling for *assigning responsibility for failure*, therefore each failure needs to be pin-pointed to one single individual.

BMW Group's culture is to avoid "failures" (in builds and in other ways) and view them as a problem, rather than a culture of experimentation to fail fast and then to learn and adapt from "failures".

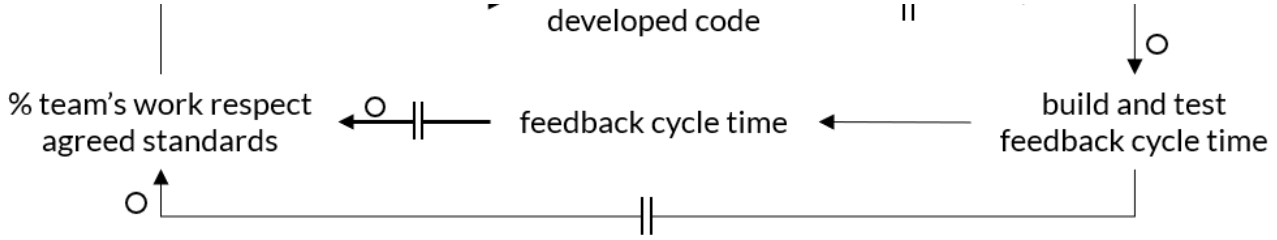


Figure 27: Dynamics of build and test cycle times to team's following standards.

Organizational Wizardry

The Quick-Fix Build Sheriff Reaction

One management reaction to the build problems was to introduce a rotating *build sheriff* (manager) role. The quick-fix of adding this role led to increasing distance and offloading of responsibility to that role. Why?

Caring about the result of a team's commit in a build is influenced by the duration of the build, and by who is responsible for problems. So, with the long build cycle and the build sheriff role, the caring and in turn the felt responsibility for the result decreased.

The decreased responsibility for build results resulted in quickly degrading builds. This was the opposite effect to that intended by the naive intervention of assigning a build manager to the task.

The effect of reduced responsibility became only apparent with growing numbers of teams. As long as there had been only about 8 teams, the not-bad social connectedness between the teams prevented this dynamic. But with more teams, the weaker social connectedness and the discussed dynamic became apparent.

Neither managers nor developers learnt that putting a manager (or a tool) between responsibility and the teams degrades the process of taking ownership and responsibility for the whole product.

Highlight Blame and the Dynamics which Follows

The line between transparency and blame is thin and easily crossed with a build monitor that highlights the particular team that pushed last before a failed build. The official phrase was "Stop and Fix assigned Team"—the designated team that broke the build (based on green-red transition on the build monitor) which lasted until the build was green again. The intent was for transparency and reducing the load of email received every day. Unofficially, all developers who cared for the build were annoyed that a broken build wasn't fixed ASAP and wished for more public pressure to fix the broken builds.

The blame-casting doesn't work on root cause (missing whole product focus), it only highlights the complexity of the system, delayed feedback from the build system, developers' focus on components rather than the product, insufficient software design, and missing meaningful component builds and tests.

The blame-casting induced an interesting other dynamic: as soon as the first team "A" (unintentionally) triggered a green-to-red transition, other teams increased their merge and push frequency. Why? First, due to long and sometimes non-linear merge processes of the auto-merge procedures on the build system, team A may not be responsible for the

transition to red. Regardless, by setting the blame on team A, other teams *pushed their commits, concealed within team A's blame*. They didn't fear being held responsible for a potential build degradation, as the build was already red. *This contributed to the practice of delaying pushes until the first team had been assigned the blame*.

By introducing a technical system to assign blame, the dynamics led to an even more occluded situation. *At times, those liberated phases of ignoring build status led to amazing collaboration across teams and code-wise integration; at other times, the whole system simply choked*.

The naive call from traditional management to simply "stop the line" until the build is restored was rarely heard. ADD's managers quickly realized that stopping all work would only degrade the situation as changes accumulate on long-living branches. In addition, prohibiting hundreds of teams to work is prohibitively expensive. The corollary to Toyota's "stop the line" philosophy: *fix the root cause*, was not realized by ADD's managers or developers.

Don't Focus on Not-Red—Focus on Green

All of the above experiments, processes, and technical aids focus on "not-red"; all of them failed. Why did they fail? They did not focus on the one thing that matters; rather, they focused on avoiding weaknesses.

Why didn't ADD strengthen "caring" and foster "ownership for the product" within the employees more?

Among other reasons, there are several major systemic root causes: (1) the C-level chosen organization setup of ADD as a department within BMW Group incurred several constraints (e.g., company mindset of outsourcing and its ramifications, company culture "excellence in details (component) not the whole (product)", procurement processes, vendor collaboration paradigms, HR policies), (2) the staff composition, the resulting attitude, and conformity to company culture, and (3) unchanged policies in HR and vendor-collaboration on all levels.

The unchanged HR policy of personalized bonus system with focus on components and specialties contradicts the required skills of large-scale, adaptive products. *Modern product development of large, complex, and adaptive systems is no longer a matter of excellence on single-focus execution but a multi-learn/skill of quick adaptation on the product level goals*. The HR managers, which are in a separate reporting line than ADD, discarded all arguments for a bonus system based on product success; they measured and rewarded local optimizations by individual managers and developers, not global optimization.

The unchanged vendor-collaboration policies led to the same uninsightful execution of thrown-over-the-wall tasks. ADD didn't acknowledge the possibilities of the changes in procurement processes, and instead just introduced a new middleman role to try to quick-fix the unskillful collaborations.

The by-design tight control from C-level managers posed another constraint. The choice to operate ADD as a department within BMW Group, would have still left options. For example, different wage agreements, independent HR policies, and more liberal processes. But the C-level managers discouraged this level of change in their organization.

Communities

Mandated and Mandatory Communities

Following "the past is part of the present" many teams demanded formal procedures for cross-team decisions. This reflects that behavioral changes in large organizations are influenced by structural changes (Larman's Law #5), but unlike the structural changes, behavioral changes aren't realized immediately. This (temporary) need gave rise to the idea of "mandated communities" and "mandatory communities". Both contradicting the idea of volunteering and definition of

Communities. To explain: mandated communities can make decisions for the whole development group (which contradicts “Communities cannot make decisions for the teams, but they can produce something that the teams decide to adopt” [3, p. 296]); mandatory communities are mandatory to attend for representatives of all development teams (which contradicts “Participation in Communities is completely voluntary” [3, p. 295]).

Why does volunteering have significance in Communities? (1) the openness for sharing and learning, high-value collaboration and contribution is higher in a volunteering (or at least not-forced) setup than a forced setup, and (2) volunteering members are more motivated to share and explain recommendations, and are more invested in adjusting recommendations to actual teams and use. The corollary is: if management feels the need to revert to a forced setup, the investments in compliance checks (e.g., conformity builds), process-supported compliance boards, explicit proclamation of new standards, etc. increase.

Please note the difference between community and Community.

Examples of mandated community: *software architecture* community and *build and integration* community; example of mandatory community: *simulation* community.

To make the voting in mandated communities “meaningful”, the concept of “core member” and quorum was introduced. To make the community meetings “meaningful”, an appointed facilitator was nominated, and prepared agendas on fixed schedules were sent out in advance. And to make the communities “consequential and meaningful” a manager was formally appointed but never present. The structure of these “communities” reflected much of the old BMW Group’s classic working group, with one change: no classical manager present. Developers’ adherence to the decisions was meager in the first years.

Why was the developers’ adherence meager? The abrupt shift from the self-management to the power-driven model with “mandated communities” conflicted with the slowly evolving adoption of self-management within the teams. The more the teams realized the freedom, responsibility, and accountability of self-management teams, the larger the resentments grew towards the command-driven “mandated communities”.

What led to the de-focusing of many developers from communities? Communities were increasingly perceived as places for formal discussions and decision-making rather than learning and achieving cross-team (functional) agreements. The lack of holistic product guidance by PO and APOs contributed to this perception. During forming and reforming of Requirement Areas, teams moved from one Area to another. On occasions, a community had members primarily from one Area. This directed topics inadvertently towards one Area’s focus, which complicated the acceptance of communities as places for cross-team and cross-Area learning and coordination. The *quick-fix of mandatory attendance* for mandated communities fixed only the issue of representation, not the root cause of missing the whole product focus in many developers.

One important secondary aspect of communities and their decisions is the ISO 9001. The process-standard ISO 9001 was adopted by BMW Group for the whole group decades ago. The basic idea of the standard is to document important process decisions and demands following these decisions. ADD’s “Coaching and Standards department” mistakenly assumed that “mandated communities” produce these kinds of decisions. Therefore, the pure existence of “mandated communities” complicated the process of quality and process improvements at ADD.

Why did ADD’s “Coaching and Standards department” make this mistake? The (by design) close similarity of managed communities with BMW Group’s traditional working groups and its formal power to decide played an important role; from an external not-LeSS-informed auditor perspective, these structures are relabeled manager-driven working groups with the lack of joining managers. Therefore, the pure existence and definition of “mandated communities” and BMW Group’s demand for ISO compliance, led to the process: some decisions of mandated communities are relevant for ISO process changes and therefore needs to be documented, tracked, and followed-up differently.

How did this mistake complicate the process of quality and process improvement? There are many aspects of how this complicated improvements: (1) the added “paper” process increased the disconnection from idea to effect (similar to the

delay-feedback effect), (2) the increased distance from improvement idea to actual improvement disconnected potential contributors, and (3) the formal process discouraged improvement experiments. All this despite the effort of the "Coaching and Standards department" to make this lightweight and as unobtrusive as possible—the pure possibility and knowledge of these processes discourage many developers from proposing improvements.

For a better frame of reference: only a fraction of the decisions and processes produced by mandated communities are relevant for ISO; improvement experiments from Retrospective and Overall Retrospective are typically not relevant for ISO process changes. For example, mandatory information in commit messages and its format, mandatory data types to represent physical units, definitions and tools for testing, folder layout and file naming conventions.

Inflation of Communities

ADD inherited from BMW Group many HR policies, especially those for bonuses and career advancements. Some of these policies influenced the number and quality of communities.

Figure 28 displays a not-so-obvious conflict between product, organization, and employee value.

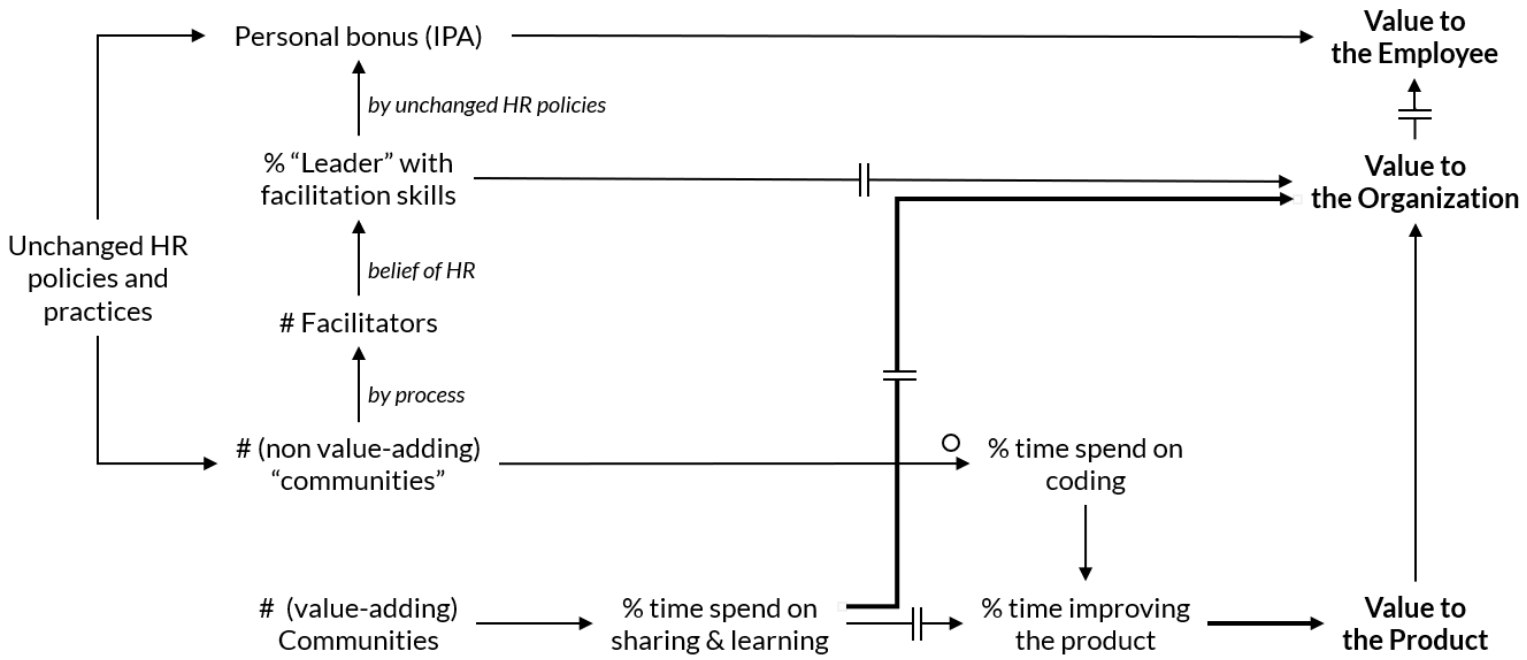


Figure 28: The Dynamics of Inflation of Communities.

Some additional context to understand the dynamics of Figure 28: in BMW Group, working groups and study groups are common. Those materialize on topics of concern and interest for the company. Typically, the facilitators are managers (or on the manager career track). So, being a facilitator has an influence on career advancement, personal bonuses, and salary level.

Within LeSS, Communities are emergent structures for learning and coordinating with by-design low barriers for creating, changing, and removing Communities. This unrestricted process in combination *with unchanged HR policies*, appraisal procedures, career advancement models, and personal bonus systems, started a dynamic leading to more and more "communities".

Why? As HR did not provide any change on career paths or bonus system some developers tried to map the old ways to the new ways. They identified communities as something close to working groups, which had been essential check-points in the old model for their career advancement.

The Quick-Fix of “Expert Teams”

ADD does not include all aspects of the product, for example, hardware architecture. So, special attention to the interfaces from the non-traditional ADD structures to the traditional BMW Group structure is required.

Hardware-related Architecture

Handling the aspect of hardware-related architecture started in the first Requirement Area with two architects from the old architecture team joining two separate regular feature teams—essentially leading teams. It was planned to form a regular Community to support learning across teams and Areas.

This approach (architect as team members) failed in this early stage due to several aspects: (1) the items for the feature teams in the early Sprints didn’t touch hardware-related architecture, and (2) the architect team members were still heavily involved in the BMW Group’s architecture group.

Why hadn’t there been hardware-related architecture items for these leading teams in the first Sprints? ADD started with vehicle prototypes and simulations before the adoption. Some details were unclear, but the big decisions were made, so the fake PO (real PO joined later) prioritized non-hardware-architecture topics, for example, software architecture-related topics and other features.

Why were the architect team members occupied if no relevant items were processed by their team? Within the BMW Group, the architecture group is a network of precursors, events, and decisions—it’s a slowly moving normative, partially political body. So, for ADD as a department to stay connected to corporate’s mainstream, close ties in the form of members joining and actively contributing is essential. This led to the architect team members spending only a fraction of their time in their actual ADD team.

ADD’s management soon placed hardware-related architecture responsibility to a dedicated “architecture expert team”. This team received added responsibility to explicitly connect with teams for sharing and learning. To clarify: this team didn’t have any obligation to implement features or work on the Product Backlog, just focusing on “hardware-related architecture”.

To contrast this with software-related architecture (i.e., software design): here, more developers involved themselves voluntarily from the start. They formed the “Software Architecture Community” which quickly transformed into a mandated community. Only after years, some core members of this mandated community were moved into the new “software architecture expert team”. This was a quick-fix for increased manager’s awareness of lacking software design capabilities in the teams and flaws in Product Backlog ordering.

One of the consequences was that most feature teams underestimated hardware-related change requests and pushed the deadlines of approval processes hard. Which then led to the gratifying learning that during Product Backlog Refinement items need skimming if hardware-related changes are needed and needs appropriate planning.

Dependability

Another “expert team” case is Dependability. Dependability in ADD’s context is safety, security, durability, maintainability, and (partially) compliance to regulatory requirements.
The content of dependability is essential for the product and should therefore be part of the Definition of Done and part of the Product Backlog Refinement. Due to the history of project thinking, component thinking, management accountability, and traditional management thinking, a dedicated group focusing on dependability existed before the start

of the adoption.

The dependability group joined late; about the time of SoP 2018 people joining. Before that, the developers considered “normal” due diligence and (softer) aspects of their Definition of Done. With the dependability group joining, the Definition of Done was revised with a gradually more complete Done.

The tasks within the dependability group before joining the adoption focused heavily among other things on tasking and coordinating suppliers for audits and studies, definition of safety goals, definition of low-level hardware requirements, coordination with regulators, and integration of regulations into safety goals. In the classical vehicle projects, those safety goals are then passed to the development managers to be considered for planning, execution and testing.

Most of the dependability groups activity before was “aside” normal team activity, therefore this aside-characteristic was first preserved. The intention is to gradually expand teams’ capability and realization of the real Done. Until this is achieved, the dependability group is part of the “undone” activity.

After joining, the dependability group stayed apart as a separate group. Like the “architecture expert team”, they received the additional duty to work closely with APO and teams during Product Backlog Refinements. Depending on the APO realization of the need, this approach only barely improved the teams’ realization of the real Done for the product.

To clarify: “Dependability Expert Team” members joined on occasion Product Backlog Refinements events and contributed to the understanding of the real Done and all the little details this requires. The automated testing suites were adjusted for inspection by “Dependability Expert Team” members, other experts, and stakeholders.

Abbreviations

Abbreviations	Meaning
ACC	Active Cruise Control
AD	Autonomous Driving
ADAS	Advanced Driver-Assistance Systems (up until and including SAE Level 2)
ADD	Automated and Autonomous Driving Department
ADP	Autonomous Driving Platform
AI	Artificial Intelligence
ASIL	Automotive Safety Integrity Level
CLP	Certified LeSS Practitioner
DCC	Dynamic Cruise Control
ECU	Electronic Control Unit
E2E	End-to-End
GHE	Git Hub Enterprise. A self-hosted version of Git Hub.
IPA	Individual Performance Appraisal
ML	Machine Learning
PO	Product Owner
SCM	Software Configuration Management
SoP	Start of Production
SWP	Software Partner
TDD	Test Driven Development
VP	Vice President

References

Ref-No	Reference
[1]	C. Larman and B. Vodde, Scaling Lean & Agile Development, Addison-Wesley, 2008. For logged-in CLPs available as PDF.
[2]	C. Larman and B. Vodde, Practices for Scaling Lean & Agile Development, Addison-Wesley, 2010. For logged-in CLPs available as PDF.
[3]	C. Larman and B. Vodde, Large-Scale Scrum, Addison-Wesley, 2016. For logged-in CLPs available as PDF.
[4]	C. Larman and B. Vodde, Go See, 2019, https://less.works/less/management/go-see.html
[5]	B. Vodde, Politics!, 2019, https://less.works/attachable/materials/916
[6]	How too many rules at work keep you from getting things done. TED Talk, 2015.
[7]	C. Larman, Applying UML and Patterns, Prentice Hall, 3rd edition, 2004.
[8]	C. Larman and B. Vodde, Lean Primer, Version 1.6, 2009. PDF.
[10]	Tuckman, Bruce (Spring 2001). Developmental Sequence in Small Groups (PDF). Group Facilitation: A Research and Applications Journal. 63 (6): 71–72. doi:10.1037/h0022100.
[11]	Adzic, Gojko (2009), “Bridging the Communication Gap” https://gojko.net/books/bridging-the-communication-gap/
[12]	J. Richard Hackman, Leading Teams, Harvard Business Review Press, 2002.
[13]	Jeffrey Pfeffer, Six Dangerous Myths About Pay, 1998, https://hbr.org/1998/05/six-dangerous-myths-about-pay
[14]	Jay R. Galbraith, The Star ModelTM, 2016, https://www.jaygalbraith.com/services/star-model
[15]	Jo Freeman, The Tyranny of Structurelessness, 1970-1973, https://www.jofreeman.com/joreen/tyranny.htm
[16]	Frederick Herzberg, One More Time: How Do You Motivate Employees?, Harvard Business Review, 2003, https://hbr.org/2003/01/one-more-time-how-do-you-motivate-employees
[17]	Alfie Kohn, Punished by Rewards, Houghton Mifflin Company, Twenty-fifth anniversary edition, 2018.
[18]	Herbert H. Meyer et al., Split Roles in Performance Appraisal, Harvard Business Review, 1965, https://hbr.org /1965/01/split-roles-in-performance-appraisal
[19]	Michele & Jim McCarthy, Richard Kasperowski, The Core Protocols, https://thecoreprotocols.org/