BHC CONSULTING

WHITEPAPER

SOFTWARE-DRIVEN PRODUCTS

– ORGANIZATIONAL WILLINGNESS AND FUNDAMENTALS OF METHODOLOGY

Management Summary

- Software-driven products are characterized by an overall increase in complexity, with this complexity shifting from mechanical elements to software.
- As the pressure to innovate is increasing, companies are being confronted with the challenge of mastering this complexity.
- The key to this is a methodical view of the overall system, for example using methods from the field of systems engineering.
- However, the human factor is just as critical to success when merging mechanics, electronics and software, as is the challenge of opening up the organization to interdisciplinary thinking.

Content

1 The world is becoming increasingly digital and software-driven - and so are its products!

Digitalization in itself is nothing new, no uncharted territory yet to be explored. Digitalization has been around for a long time, and it is here to stay. Although the interconnection of things is proceeding rapidly, only the surface of the possibilities it offers has been scratched. This can also be seen in our day-to-day lives, and it might well be that in five years' time no one will remember that a bicycle was once a purely mechanical product.

The driver and enabler of this development is software. Increasingly, the functionality of a product no longer derives from its electromechanical qualities alone, but rather from an ever closer symbiosis between software and hardware, be it in cars, in medical technology, in mechanical and plant engineering – or even in bicycles.

From a business perspective, these developments represent both an opportunity and a challenge.

An opportunity because, given increased competitive and innovative pressure, software-driven products make it possible to up the tempo and reduce development times, but also open the way to completely new business models. Take e-bikes, for example. Because sensors already control pedal assist and the display is connected to your mobile phone via Bluetooth, it is just a short step to cloud-based measurement of training success. Or how about a power boost if the hill is again just a bit too steep – for a small charge paid by mobile phone of course?

A challenge because this change is anything but trivial. Even if variance may decrease on the mechanical side, the complexity of the overall system will increase. If only because behind the hardware and software there are often different development teams, or at least interdisciplinary groups of people with different ways of thinking, who drive the innovation cycles in their discipline at different speeds.

Thought must also be given to how the overall system can be validated – especially in the context of safety-critical products. Speaking of which, regulations may require that it be possible to trace every development step and every individual decision right back to the definition of requirements, depending on the industry and product in question (more on this in the BHC white paper „Software-driven products – Establish an End-to-end traceability"). When it comes to increasing complexity, most companies are likely to have similar experience when addressing the issue of merging hardware and software for the first time – if not earlier.

Even though overall product complexity will increase in the future, a large proportion of the complexity will shift from the electromechanical side to the software. Although that alone will not make things any easier, removing the constraints imposed by the physical world will allow the application of other methods for mastering complexity, with a disproportionately higher level of efficiency and scalability. Those who master this discipline will therefore be able to boost their performance to a far greater degree than the degree to which the effort required to master complexity increases. Unfortunately, however, it is also true that this is easier said than done; and those who do not manage to do this will sooner or later be overwhelmed by the complexity.

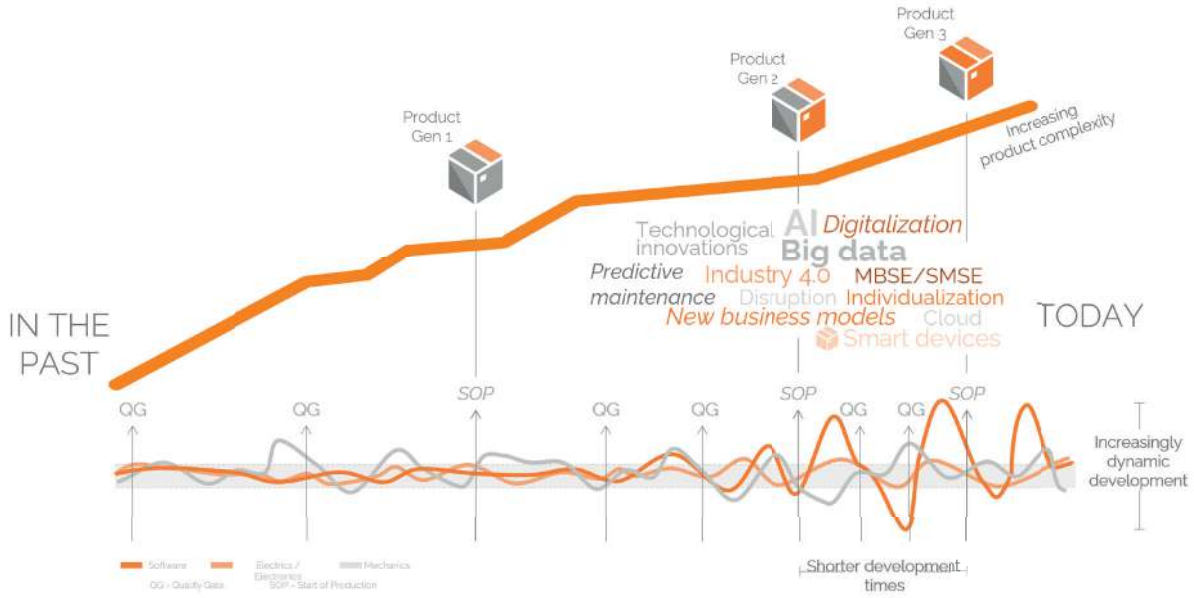


Figure 1: Increasing product complexity and ever shorter development times

2 Why you need to take a new look at your lifecycle management system for software-driven products

The different histories of conventional engineering and software development have led to two different and specialized worlds of tool and methods: PLM and ALM

The functional symbiosis between mechanical engineering, electronics and software and a shift of functional complexity toward software is made possible by a steady increase in computing power. However, this also has an impact on the scope of the software, and wherever there is interaction between a large number of elements, there is, statistically speaking, unfortunately also an increased probability that errors will occur. By way of comparison: a Linux kernel from 1994 required just under 200,000 lines of code; by 2018, that figure had risen to over 25 million. To make matters worse, a lack of structure reduces the efficiency of troubleshooting. Over time, the methods toolbox used for software development evolved into what we know today as application lifecycle management (ALM), and its process model has been incorporated into modern systems engineering.

In the world dominated by electromechanical elements, on the other hand, the need to support the design process, manage the technical data for components, and the ability to work together as a team is the main issue. In many companies, focus was and still is placed on managing the many individual components, while ensuring consistency from the requirements specification through to the finished product is not usually supported by a product lifecycle management (PLM) concept (method+tool) that has been established consistently throughout the company. Even if state-of-the-art PLM solutions today were able to do much more than they are currently used for, the strengths and focus of use still reside in a large number of disciplines that have simply no relevance for a pure software product: 3D-based design, mechanical calculations, assembly planning, and support for manufacturing, procurement, and logistics processes in the supply chain.

Neither parallel PLM and ALM worlds nor half-hearted integrations do justice to software-driven products

A company that has grown up with purely mechanical or electromechanical products and is now increasingly integrating software will struggle with the challenge that the existing sets of tools and methods come primarily from the world of PDM/PLM. However, the complexity of software products is not simply added to the existing complexity. Rather, the overall complexity is increasing exponentially. Ultimately, the number of participants, the number of process-related interfaces and the need for coordination among participants, whose ways of working and thinking are very different, all increase. The requirements that have to be fulfilled do not become less but rather more demanding and, in particular with regard to safety aspects (in the sense of functional safety), also more stringent.

In our customer projects, we observe that companies often respond to the emergence of software in products with one of the following approaches, which we present in simplified form:

- Either the **software is seen as a hardware appendage**
  (along the lines of "that little bit of software is just another part number")

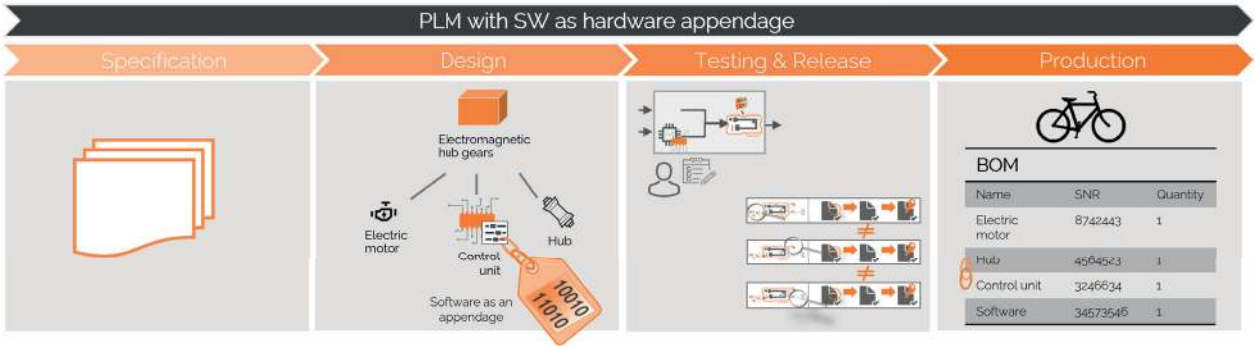


Figure 2: Software as a hardware appendage

Software components are equated with electromechanical components and are assigned a part number. While you may still be able to identify at least the ECU in a product structure, it will ultimately be impossible to identify all the mutual dependencies in a flat BOM.

- Or **an independent parallel ALM world is set up alongside the PLM world** (along the lines of "I don't know what they're doing over there, but I'm not interested either")

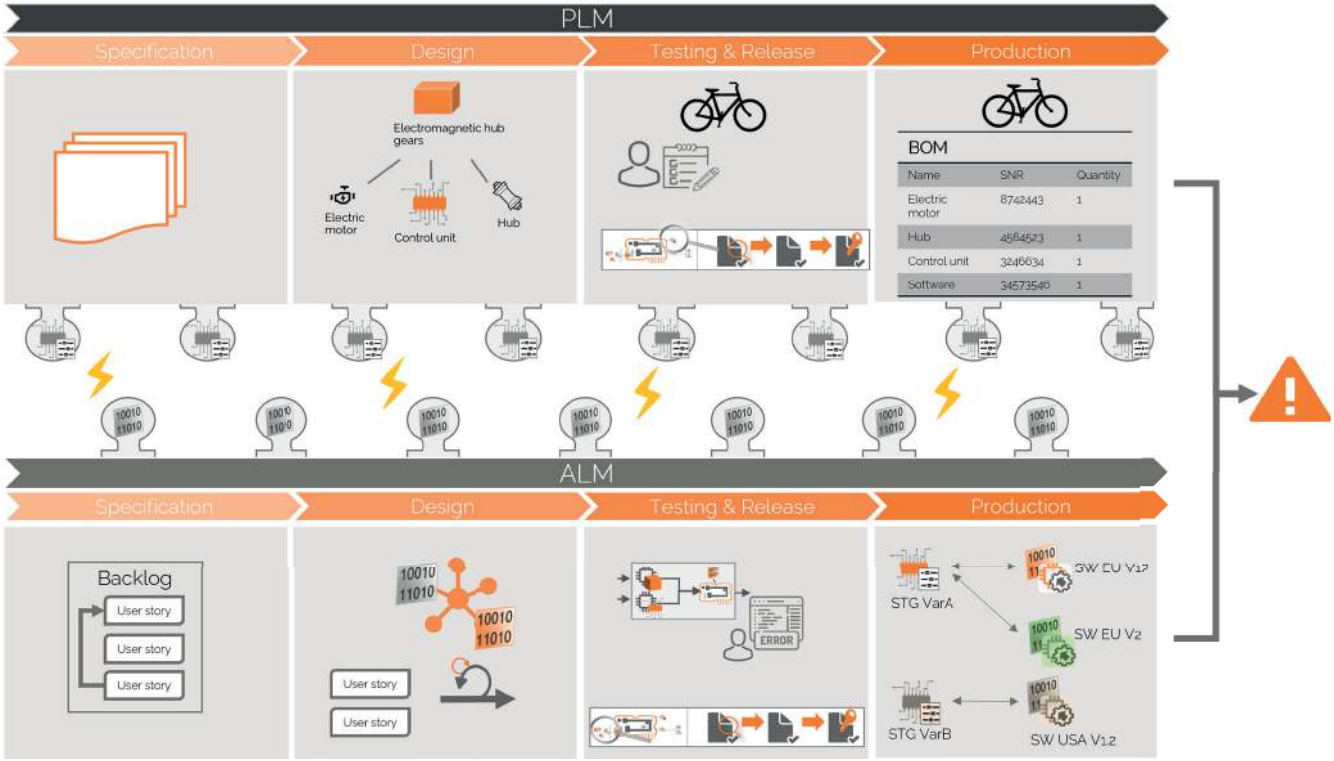


Figure 3: Parallel PLM and ALM processes

Freed from the fetters of electromechanical development, the software developers can give free rein to their dynamic capabilities in their ALM world. The software is optimized to meet current customer requirements in short iterations and in a very agile manner. The thing that is usually overlooked in a scenario of complete separation is mutual synchronization, and this then becomes an inconsistency that is carried over into production and the finished product.

It is extremely rare that either approach is ideal. Treating software as a mere appendage to hardware may still work if the level of functional integration is not particularly high or the product is not subject to high rates of change. In other words, if the software is there to solve specific and clearly delineated and localized tasks and its impact on the overall system is not significant. In addition, there should be no great expectations with regard to the agility of software development.

The second approach may be correct purely from the perspective of the software, but not in a holistic view of the product and its value creation processes. Ensuring interaction and validation across the boundaries of different tools and departments may then become a challenge that cannot be solved.

Depending on the complexity of the variance in a product and the relative size of the company, both strategies can work at least for a while. Fortunately, it is often enough the case that methodological and procedural deficiencies are compensated for by motivated and competent employees. Moreover, companies quickly develop a remarkable level of tolerance and capacity for suffering in the face of systemic shortcomings. This can escalate into an increased willingness to take risks, which results in inadequately validated products being brought to market. Customers will also accept a certain level of imperfection in the product for a while until they can no longer identify any added value compared with competitor products and simply disappear into the distance.

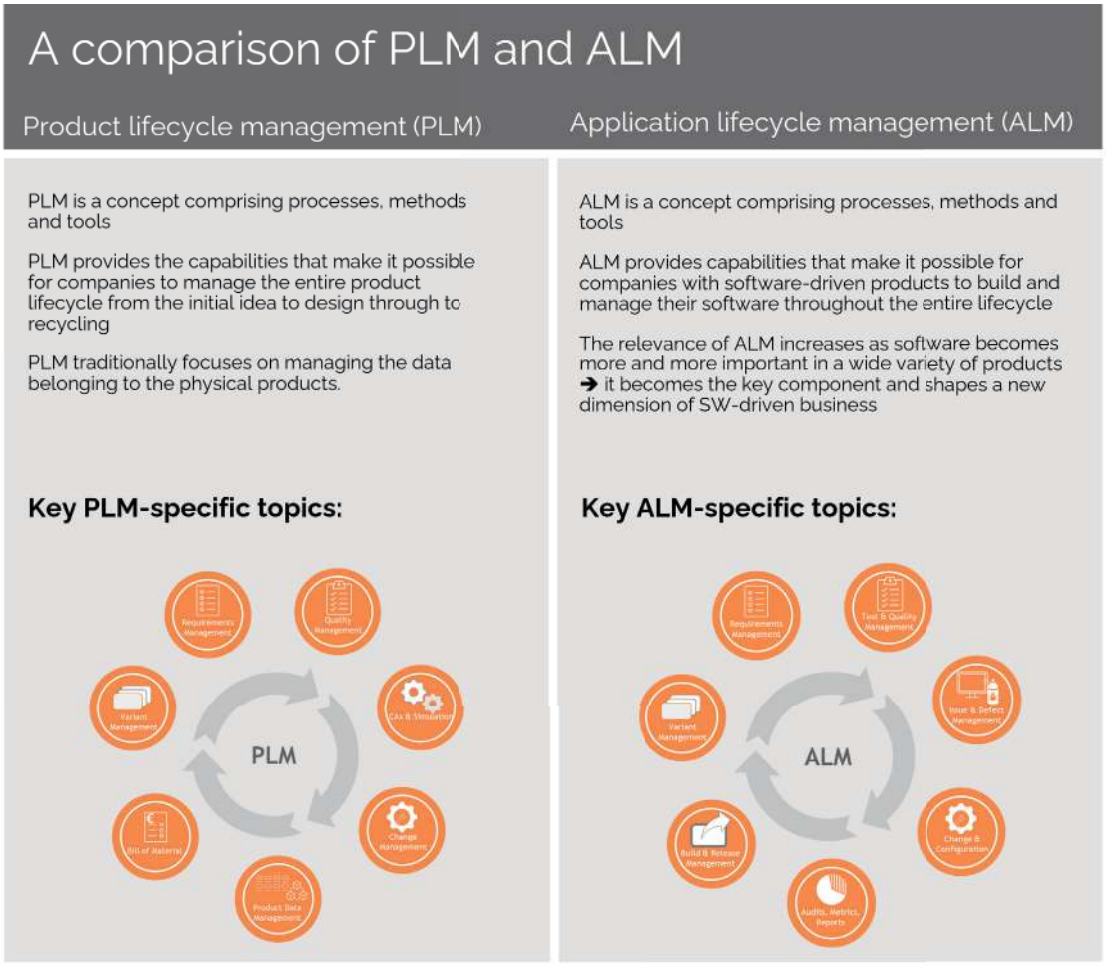


Figure 4: Comparison of PLM and ALM

3 Rethinking PLM and ALM – getting to grips with increasing complexity

The „fine art“ therefore lies in making available processes, methods and tools that give all the domains involved in product development the space they need to flourish. At the same time, however, they have to be orchestrated and coordinated to ensure that the end product meets all the requirements and functions as a single unit. This is not primarily merely a question of tools, processes and methods but instead depends first and foremost on whether a company is ready to make sweeping changes.

Willingness on the part of the organization

Unfortunately, the human factor is far too often underestimated when it comes time to make fundamental changes. And companies are made up of people. The challenge of developing a software-driven product may seem small to a young startup. But it can become an almost insurmountable obstacle for an established company with ingrained structures and practices. The problem is often not even the employees' technical expertise but rather managing and orchestrating the whole ensemble or the organization's lack of willingness to be orchestrated.

In practice, experience shows that this is one of the greatest challenges – this often means that projects and initiatives are launched with great enthusiasm but ultimately do not lead to the desired result. And the more ambitious and extensive a project, the more often this is the case.

Merging hardware and software in a single product in an agile and reliable manner requires all participants to change their way of thinking. It is like two groups of friends with different hobbies wanting to do more together – one group of friends are enthusiastic hikers, while the other prefers to go biking. One or the other hiker might even imagine that if they all want to do go on a trip together, the bikers can simply join them on their next hike. After all, anyone can hike but not everyone is keen on riding a bike. But that is, of course, not a long-term solution because it means that the bikers simply become “appendages” (along the same lines as software as an

appendage). A solution that would work for everybody is, however, close at hand. Who says that a group trip has to mean everyone going on a hike? Getting together could also mean meeting up half way and enjoying a picnic together and meeting up again in the evening for a bite to eat. Finding this solution, however, requires that everyone involved think outside the box and be willing to look at things from a different perspective.

Companies, however, are not usually dealing something as uncomplicated as a group trip. Instead, they are dealing with departments comprising dozens or even hundreds of people. The more heterogeneous the environment, the more important it is to hold this diverse group of people together and lead them toward the shared objective. In an orchestra, this is done by the conductor, on a building site, the architect in their role as site manager. In companies, on the other hand, this important coordinating function tends to be an afterthought or is carried out by people who do not know enough about interdisciplinary collaboration.

We therefore recommend that you actively oversee the changes and anchor coordination in your organization with a clear and strong mandate. Do not merely have someone performing this task on the side as a kind of hobby (someone who might actually have been the last person to volunteer). Choose people who are well respected and do not place too much focus on one particular area of expertise and who have, or are willing to acquire, a healthy mix of pragmatism and methods expertise.

Methodological foundation

Once there is a willingness to accept new ways of thinking and working, the next step is to establish a joint procedural model for developing solutions.

Let's return to the two groups of friends and assume they have „opened their minds" and now want to find a solution for the next trip that both the bikers and the hikers will enjoy equally.

During an impromptu web meeting, both groups decide that the trip should be to the Black Forest, and they also pick an appropriate restaurant together. Finding a suitable spot for the picnic is a little trickier – after all, it has to be easy for both the hikers and the bikers to reach and it needs to meet their respective needs. This synchronization point is also something that the friends have to decide on together. Once this has been done, each subgroup can make the most of it and plan their ideal hiking or biking route. Julia, who belongs to the group of bikers and works as risk manager, mentions that some of the trails in the area are often closed, which might make it difficult for everyone to arrive at the picnic spot at the same time. To minimize the risk of this happening, they all agree that the two groups should call each other at around the halfway point so that they can compare their estimated times of arrival and modify their routes if necessary. They also make sure to pack some card games just in case, so that no one will get bored if they arrive early.

This type of planning, which might work intuitively in smaller private circles, needs to be actively controlled in the complex world of corporate reality. Methods used in systems engineering provide a suitable foundation. These already include extremely useful toolkits that can be used to gear all the components of a product or system to the shared requirements. Whether this involves you implementing one of the systems engineering standards exactly as specified or merely using it as a guideline is almost a matter of preference – unless, of course, you have to provide proof of compliance with specified standards to your customers or other stakeholders (as required in some industries), in which case, it is truly important.

What is crucial is that your company internalizes an appropriate procedural model such as the V-model and uses it much like you would a compass. The approach is not unlike the one taken when planning the private trip:

You describe all the requirements of the integrated product (an trip that includes a picnic and is neatly rounded off in a leisurely manner), give thought to the functional architecture (journey, outdoor exercise, stops for something to eat), determine who makes which functional contribution (people who have bike racks on their cars do the driving, the others organize the picnic), identify risks (closed hiking/bike trails) and determine the integrative validation levels (one group calling the other at a certain time). Apart from that, each sub-discipline has all the freedom they need to play to its strengths (hikers: direct route with some rocky stretches, bikers: long, roundabout route with flat trails).

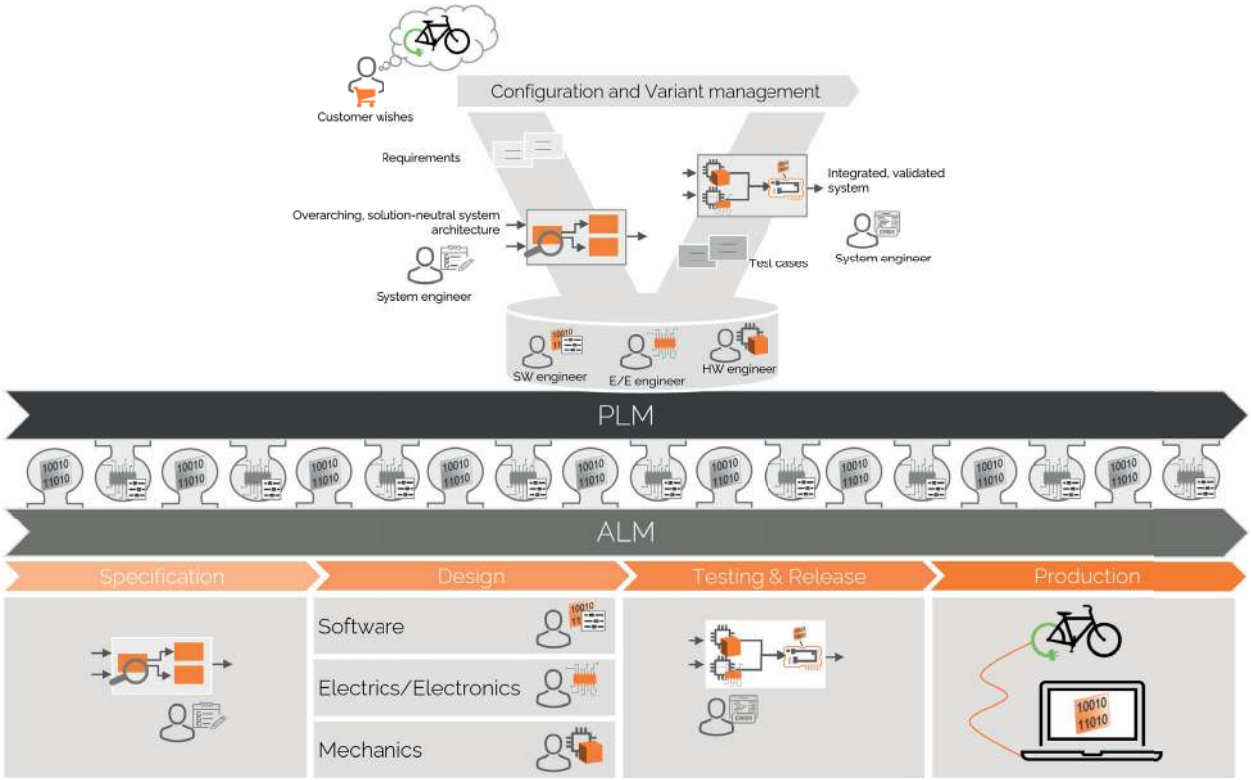


Figure 5: Systems engineering V-model for synchronizing the domains

What does this mean in the context of a software-driven product? It is vital at the very beginning that you think about what the product should be able to do and what other requirements (e.g. standards) it has to meet. This should be done as impartially as possible and without a specific approach in mind.

Describe what your product should be able to do

Once the requirements pertaining to your product are clear, the next step is to determine the functionality that each individual sub-discipline (mechanics, electronics, software) will contribute. You are still in the phase in which all participants need to work together closely. Do not, however, succumb to the temptation of wanting to specify everything down to the last detail. Take an e-bike as an example: In order to satisfy a requirement regarding an „electronic bike lock" function, all you need to specify at this stage is that there has to be some kind of mechanical locking mechanism that can be operated via software using the display on the bike. How this is actually implemented is, however, not yet relevant.

Design your systems in a way that makes sense

There are two important aspects to the architecture phase: to divide the system up sensibly and to provide an initial abstract definition of the interdependencies between the system elements. In this context, "sensible" means that the dependencies between the subsystems should be kept to a minimum, because all coordination between the individual development teams in your company will from this point on revolve around these dependencies. As the project progresses, it is then important to describe the interfaces in ever-increasing detail until finally the product is completely defined (or defined as an MVP).
Beyond initial development, the way in which the system is divided up and the description of the dependencies are also important for ongoing product maintenance. This approach allows the individual disciplines to flourish and drive innovations agilely at their own pace, provided that none of the limits imposed by the dependencies of the subsystems are exceeded. As far as a software-driven product is concerned, this means that the possibilities on the software side are boundless provided that the hardware and mechanics do not impose any constraints.

If more comprehensive further developments are involved, multiple subsystems and disciplines almost always have to be taken into account because software alone can no longer be used to implement every new requirement; changes will also have to be made to the hardware. That is why your activities should include a development roadmap that indicates which major functional innovations and which extensions to the interfaces of the subsystems are planned.

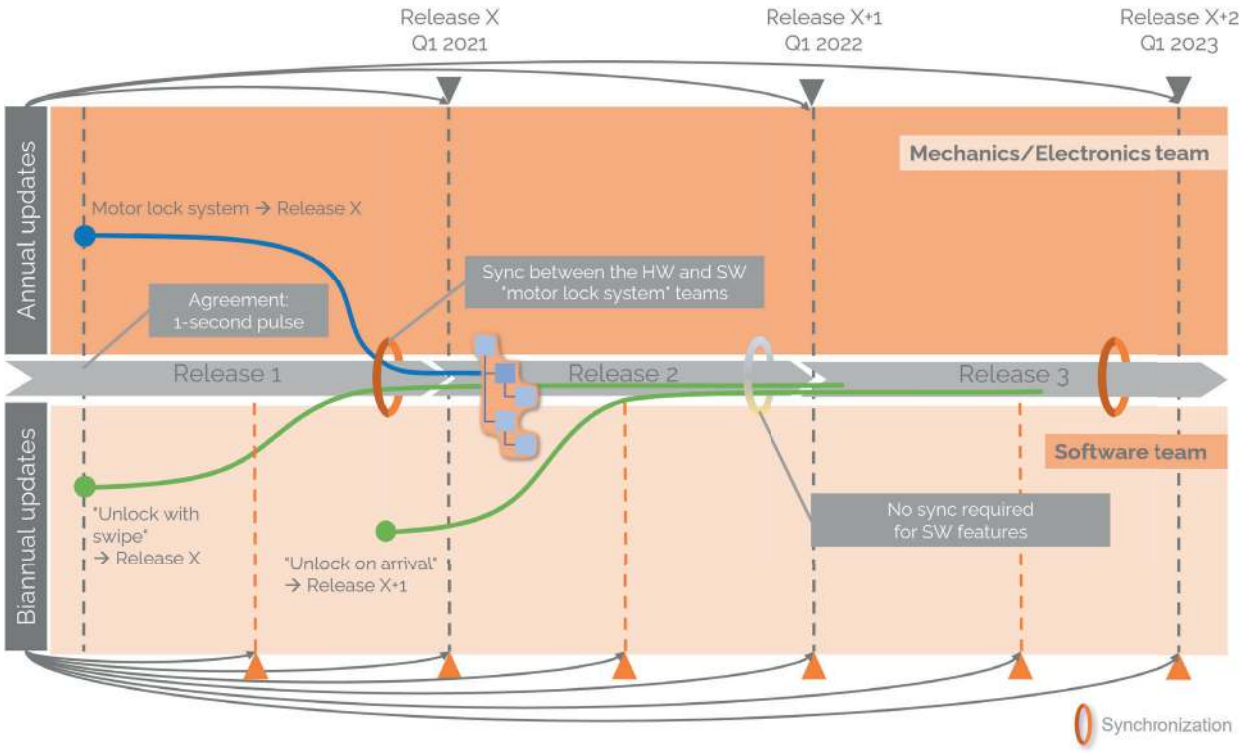


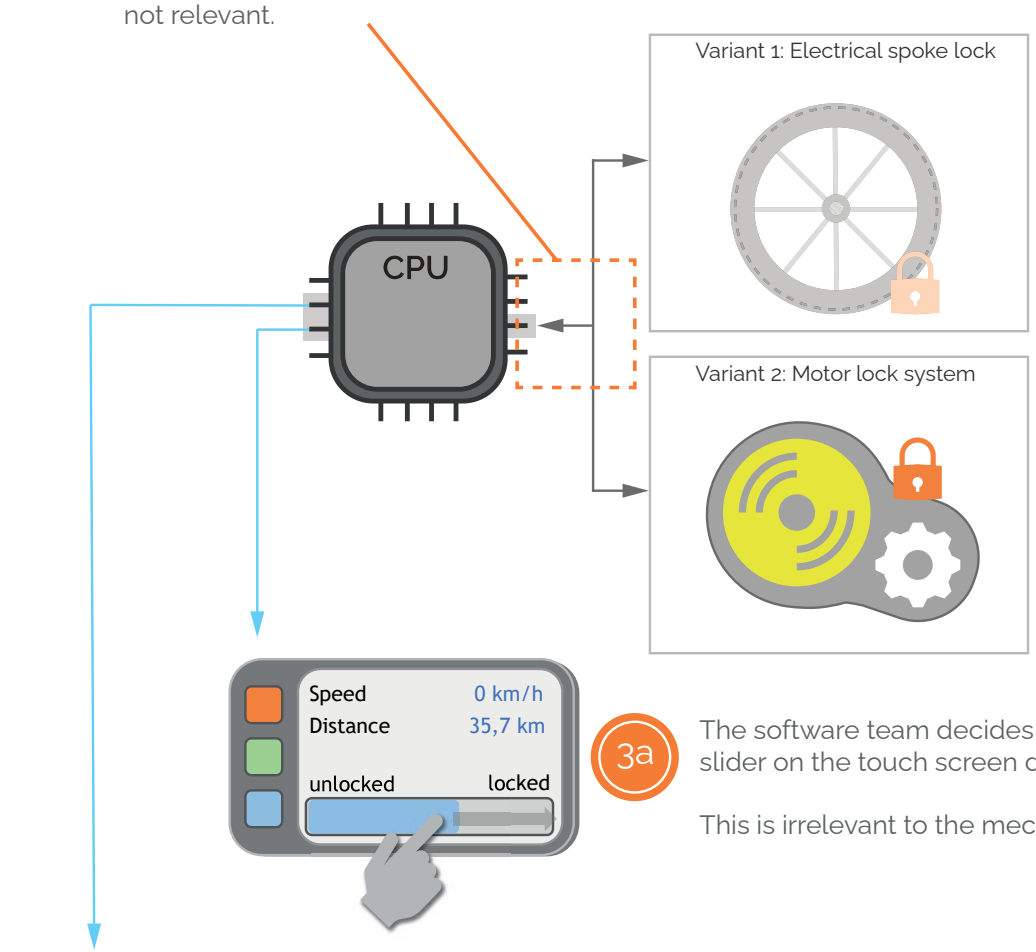
Figure 6: The different domains "software" and "mechanics/electronics" are synchronized, e.g. via synchronization points on a common roadmap

The overall system sets the pace and all the subsystems involved have to follow. For example, an e-bike manufacturer could bring an updated model to market every year. The development teams have to implement the main features planned for these annual updates in a timely manner, which of course means that the interfaces between the subsystems also need to be defined in the context of the architecture. The electronic bicycle lock function, for example, could look like this:

1

The software and mechanics/electronics teams agree to develop an electrical interface on a control unit that can be used to change and check the locked status.

There is no need to agree on technical implementation of the locking function as this is not relevant.



2

During development, the mechanics/electronics team examines the use of an electric spoke lock or a motor lock system and decides in favor the latter.

This is, however, irrelevant to the software team.

3a

The software team decides to use a virtual slider on the touch screen display.

This is irrelevant to the mechanics team.

3b

A later SW version includes a locking function that will also depend on whether or not the owner's mobile phone is nearby.

This further development is also irrelevant to the mechanics team

Figure 7: Example for the coordination of the "mechanics/electronics" and "software" teams

Otherwise, the two subsystems would develop independently of each other at different speeds. This means, for example, that a basic variant and a revamp of the mechanics/electronics could be planned for each model year, while new software versions are perhaps released in an agile manner monthly or at even shorter intervals.

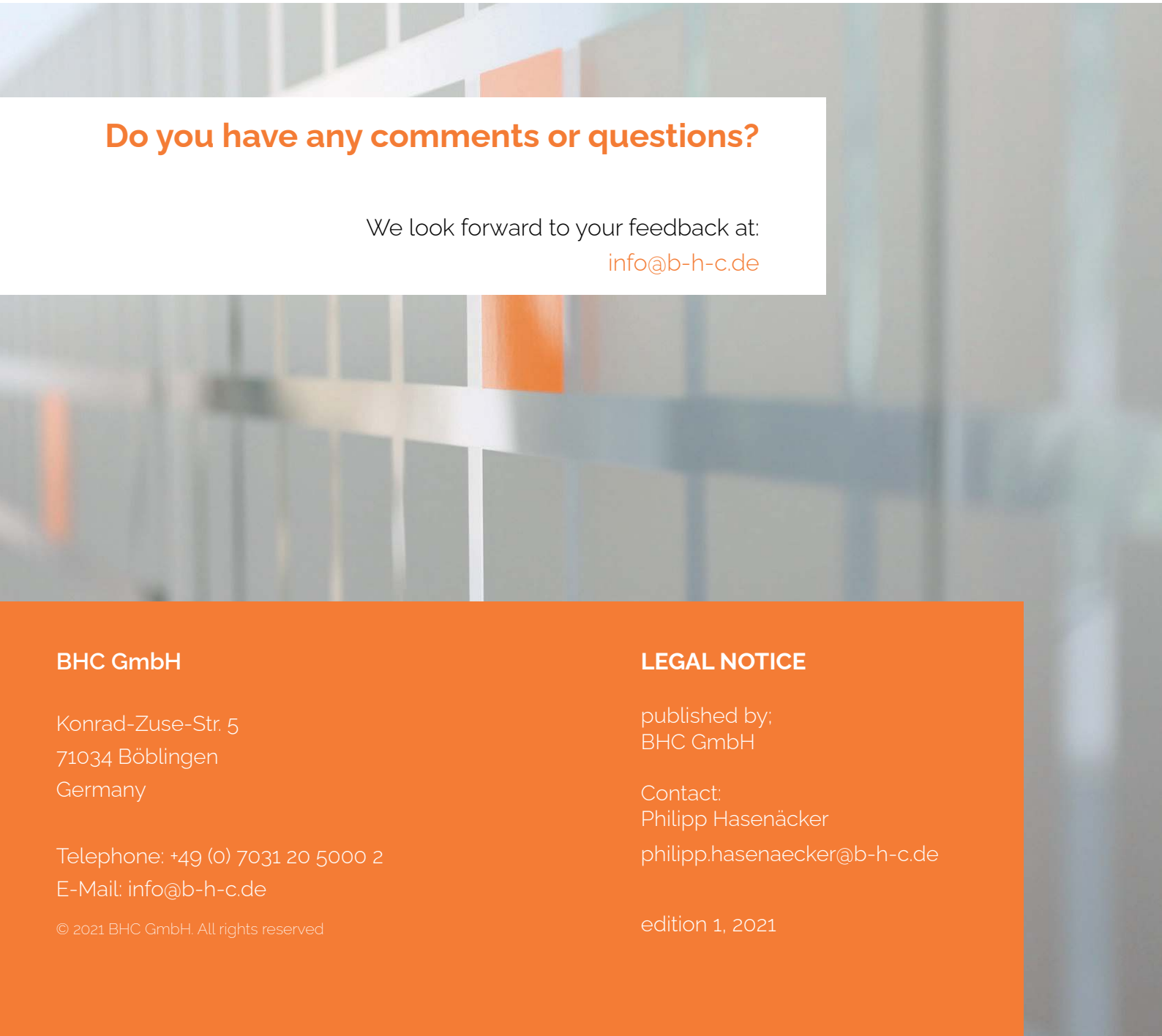
Dividing the system into the subsystems mechanics/electronics and software is of course only an example. More complex products may involve many other subsystems and possibly even several parallel subsystems of the same type (e.g. several software subsystems). It is important to keep in mind that despite every effort to keep things simple, the interdependencies between these subsystems can quickly become very diverse and complex. It is therefore essential that your IT landscape helps you keep track of all these interdependencies as best possible (more on this topic in the BHC white paper „Software-driven products: Crucial capabilities for your PLM/ALM-tool-landscape“).

Be that as it may, establishing this type of method model lays a foundation that will enable you to meet the challenge of developing software-driven products reliably throughout the development process.

4 Summary

In the future, many products will be driven by software to a far greater degree than today. Although this means that product complexity will increase, methods exist that allow this complexity to be managed reliably. End-to-end systems engineering in particular provides crucial support. But this presupposes that companies establish a culture of change, are open to interdisciplinary thinking and are prepared to throw old habits overboard. Hardly anyone need be afraid of the unknown as most companies today are already putting much of this into practice in one way or another. There is often simply a lack of optimized and more target-oriented orchestration.

BHC GmbH helps focus on what is important and brings with them methodological know-how and the experience gained in a wide-ranging portfolio of similar projects.



Do you have any comments or questions?

We look forward to your feedback at:
info@b-h-c.de

BHC GmbH

Konrad-Zuse-Str. 5
71034 Böblingen
Germany

Telephone: +49 (0) 7031 20 5000 2
E-Mail: info@b-h-c.de

LEGAL NOTICE

published by:
BHC GmbH

Contact:
Philipp Hasenäcker
philipp.hasenaecker@b-h-c.de

edition 1, 2021