ptc

DIGITAL TRANSFORMS PHYSICAL

BHC CONSULTING

Integrating Hardware and Software Development in

Digital Product Delivery

Introduction

Every day, software becomes more crucial to the way our world works – and the more we incorporate it into our daily lives and the things we use, the more complex it becomes.

To put that into perspective, the Apollo 11 moonshot required approximately 145,000 lines of code back in 1969. Today, it can take up to 100 million lines of code to get a modern car out of the driveway.

The more software that products contain, the more complex it becomes to develop them – and the more room there is for error. Meanwhile, the pressure to innovate and bring complex quality products to market faster is on more than ever. Add to that the challenge of efficiently managing the parallel development streams of hardware, software, and service innovation, ensuring transparency, and integrating all of these in a single product.

Manufacturers who don't want to get left behind in the race to optimize complex product development need to significantly evolve their processes, systems, and team mindsets, or be replaced by competitors who have designed their businesses with complex products in mind from the ground up.

The key to this evolution:

Establishing a methodical view of the overall system

Using methods from systems engineering

Opening up the organization to interdisciplinary thinking

Leveraging the right PLM/ALM tools

Software opening up new perspectives for the manufacturing business

Over 100 years after Henry Ford disrupted the automotive sector by introducing moving assembly lines, the manufacturing industry is facing radical disruption again.

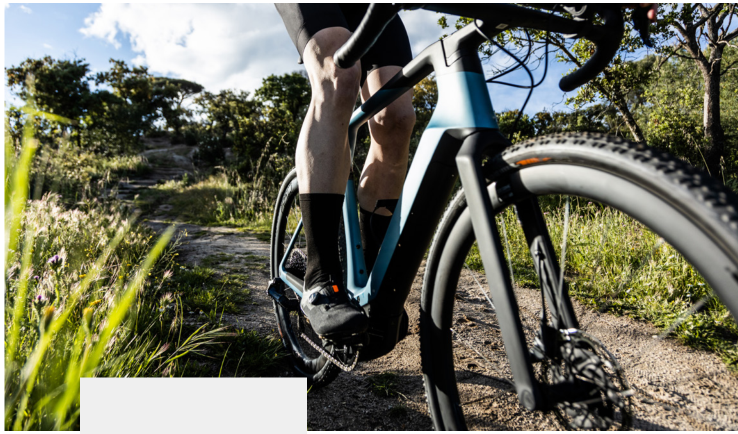
But this time, rather than a single innovation, this change is due to a combination of evolving business models, technological innovation, and supply chain changes.

The driver and enabler of this development is software. The functionality of a product is no longer down to its electromechanical qualities alone, but rather, it comes from an increasingly closer symbiosis between software and hardware, whether it's in cars, 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 competition as well as pressure to innovate, software-driven products make it possible to speed up development times as well as pave the way for completely new business models.

Take e-bikes, for example. Sensors already control pedal assist and the display is already connected to your mobile phone via Bluetooth. As a result, it's just a short step from the status quo to imagining a cloud-based performance measurement system that tracks your training efforts and offers personalized training advice. Or even a handy power boost if a hill gets a little too steep – for a small charge you can pay for by mobile, of course.

So what makes it a challenge?



Interdisciplinary teams

Well, from the product developer's perspective, this type of change is anything but straightforward. Even if variance decreases on the mechanical side, the complexity of the overall system increases because of all that software, and the tight interrelation between these components. And on top of that, there are often different development teams behind the software itself (or at least interdisciplinary groups of people with different ways of thinking) that all manage innovation cycles in their own way and at their own speed. Orchestrating their work and integrating these parallel development streams is a challenge.

System validation

Manufacturers must also consider how the overall system can be validated, especially in the context of safety-critical products. Depending on the industry and product in question, standards require every development step and change to be traceable back to the original requirements.

That being said, although the overall product complexity will increase in the future, a large part of that complexity will shift from the electromechanical side to software. Although it doesn't exactly make things easier, removing the constraints imposed by the physical world makes it possible to apply other methods of mastering complexity, with a disproportionately higher level of efficiency and scalability.

This means that those who go the extra mile to master this discipline will be able to boost their performance to a far greater degree than the effort required to master complexity.

Of course, this is easier said than done, so let's explore the kind of tooling that will support these change management efforts!

Tooling questions of managing software-driven product lifecycles

The functional symbiosis between mechanical engineering, electronics, and increasingly complex software is made possible by ever-growing computing power.

However, this also has a downside. Whenever a large number of elements interact, there is also, statistically speaking, 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 toolkits 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 many companies, the focus was and still is on managing the many individual components. Ensuring consistency from the requirements specification through to the finished product is not usually supported by a Product Lifecycle Management (PLM) concept (method & tooling) that has been established consistently throughout the company.

In a world dominated by electromechanical elements, the main challenges are as follows:

- The need to support the design process
- Managing the technical data for various components
- The ability to work together effectively as a team.

Common approaches to developing software-driven products

We can observe the following tendencies when it comes to integrating growing amounts of software in companies that grew up focusing mainly on mechanical and electromechanical products:

A

Treating software as a hardware appendage

This is when the software is seen as an extension of or addition to hardware (something along the lines of "that little bit of software is just another part number").

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.

B

ALM and PLM living side by side

In some cases, an independent parallel ALM world is set up alongside the PLM world. This presents a situation best described as "I don't know what they're doing over there, but I'm not interested either".

Freed from the constraints of electromechanical development, the software developers can fully embrace their dynamic capabilities in the ALM world. Software is optimized to meet current customer requirements in short iterations and in a very agile manner.

However, what is usually overlooked in a scenario of complete separation is mutual synchronization between the worlds of PLM and ALM. Unfortunately, this becomes an inconsistency that is carried over into production and the finished product.

The advantages and disadvantages to both approaches

Neither scenario outlined above is ideal.

Treating software as an “add-on” to hardware can work if the level of functional integration is not particularly high, or alternatively if the product doesn’t experience high rates of change. In other words, if the software is there to solve specific, clearly outlined, and localized tasks, then its impact on the overall system is not significant. In addition, there should be no great expectations in terms of the agility of software development.

The second approach can work from the software perspective, but not in terms of creating a holistic view of the product and its value creation processes. Ensuring efficient collaboration and validation across various tools and departments, in this case, continues to be a problem.

Depending on the complexity of the product variance and the relative size of the company, both strategies can work for a little while. Motivated employees often compensate for methodological and procedural shortcomings, and companies quickly develop a remarkable level of tolerance of systemic issues.

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 (at least for a while) until they can no longer see a reason not to go with a competitor instead.

Rethinking PLM and ALM: getting to grips with increasing complexity

The "fine art" of optimizing software-driven product development lies in establishing processes, methods, and tools that give all involved parties transparency, an efficient hub to collaborate, and all the tools they need to flourish.

That being said, all the separate domains still have to be coordinated to ensure that the end product meets all the requirements and functions as a single unit. However, this is not just a question of tooling and methodology. It also requires a wide-spreading and deep organizational change, the willingness for which is crucial. It is recommended to assign someone to take an active role in overseeing the changes and provide strong guidelines for coordination in your organization.

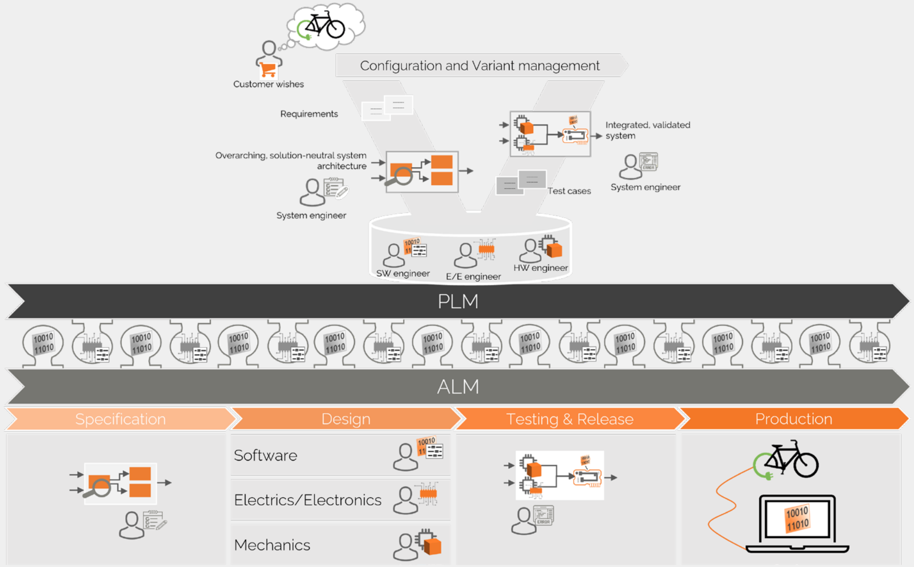
The methodological foundation

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

Impromptu planning and coordination (with sporadic and haphazard interactions between different disciplines) just won't cut it. The collaborative development process needs to be actively controlled in the complex environment of technology development.

Methods used in systems engineering provide a suitable foundation. These already include extremely useful toolkits that can be used to tweak all the components of a product or system to the shared requirements.

Whether this involves 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 becomes highly important.



Using an appropriate procedural model

What is crucial is that your company chooses an appropriate procedural model (such as the V-model, for example) and uses it much as you would a compass.

What does this mean in the context of a software-driven product? It is vital that at the very beginning, 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. Let's explore the specific steps which follow.

Describing 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.

Designing your systems in a way that makes sense

There are two important aspects to the architecture phase: first, to divide the system up sensibly, and second, 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 important to describe the interfaces in ever greater detail until the product is completely defined (or defined as an MVP).

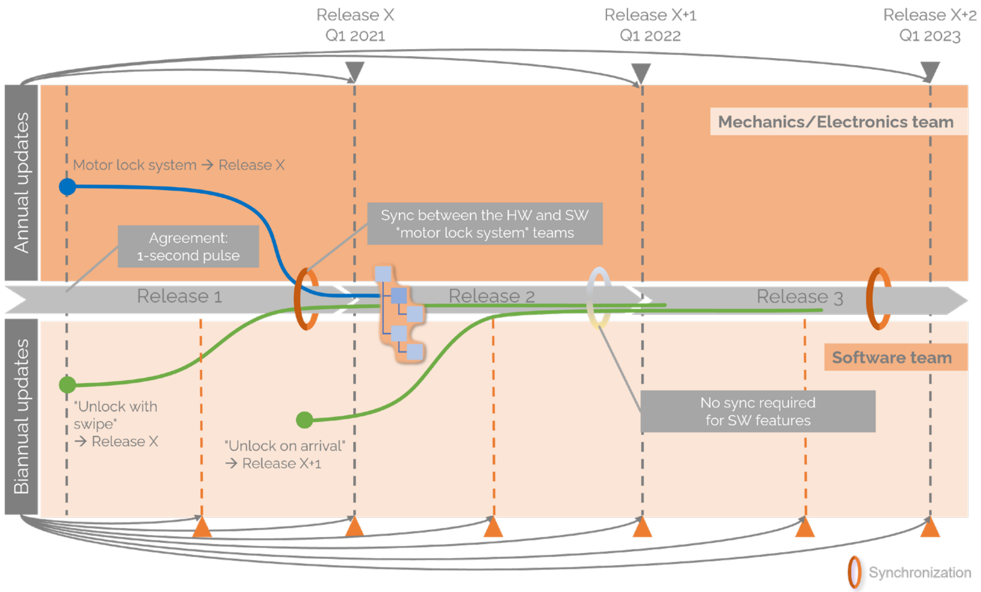
Beyond initial development, the way in which the system is divided up and the description of the dependencies are also important for ongoing product maintenance. This approach allows the individual disciplines to flourish and drive innovations agilely at their own pace, provided that none of the limits 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.

Creating a synchronization mechanism

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. In other words, 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.

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 good time, 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.

Otherwise, the two subsystems would develop independently of each other at different paces. 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 can be released monthly in an agile manner, or at even shorter intervals.



Closing thoughts & Summary

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.

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.

In the future, many products will be driven by software to a far greater degree than today.

Although this means that product complexity will inevitably increase, methods exist that allow this complexity to be managed reliably. End-to-end systems engineering, in particular, provides crucial support.

But this presumes that companies establish a culture of change, are open to interdisciplinary thinking, and are prepared to throw out old habits. Hardly anyone need be afraid of the unknown as most companies today are already putting much of these concepts into practice in one way or another. There is often simply a lack of optimized and more target-oriented coordination, which can be solved with the right mindset shift, by putting someone specific in charge, and by using adequate tooling to support your effort.

BHC CONSULTING

The BHC GmbH, as part of PROSTEP AG is specialized in IT-related consulting for product- and application lifecycle management in the automotive industry, mechanical engineering and plant engineering. The focus of our competence lies in the consulting of consistent process, method and IT-system development for companies in the fields of mechatronics and software development. To learn more, follow BHC GmbH on Linkedin.

codebeamer™

Codebeamer offers industry-leading software tools to simplify complex product and software engineering at scale.

Our enterprise-grade platforms help accelerate the development of technology products and simplify regulatory compliance. PTC's solutions are used by leading companies including top automotive, medical, pharma, and life sciences developers worldwide to manage their innovative, compliant product engineering processes.

Learn more

TÜV NORD
TÜV NORD Systems GmbH & Co. KG
Trusted Tool

ptc

DIGITAL TRANSFORMS PHYSICAL

121 Seaport Blvd, Boston, MA 02210 : ptc.com

020-integrating-hardware-software-development-digital-product-delivery-02-13