NOBLEO TECHNOLOGY

LET'S OUTSMART TOMORROW

The benefits of system engineering methodologies and tools in designing a smart and effective energy storage system

B.A.G. de Wit, M. Talebi, F. Sperling,
H. Geurink , D. Dombeeck

Februari 2025

The benefits of system engineering methodologies and tools in designing a smart and effective energy storage system

B.A.G. de Wit[1], M. Talebi[1], F. Sperling[1], H. Geurink[2] , D. Dombeeck[2] ǂ

February 2025

Abstract

An inevitable feature of renewable energy is its unpredictable availability. Thus, the global move towards renewable energy means that there is a huge demand for means to store energy on a large scale (MWh) in a smart manner [1]. The DNV's Energy Transition Outlook of 2023 has predicted over 4,500 cumulative GWh of stationary energy deployments by 2050 [2]. However, So far the slow response in the battery market has meant a growing gap between supply and demand, and suitable alternatives in the market still seem few and far between. In this article, we identify challenges in design and development of such Energy Storage Systems (ESSs) and by using a case study, describe a methodology to tackle these challenges in an accelerated effective manner. By combining the PLCnext Technology with the Nobleo Engineering Methods, we show how to create an ESS which can be seamlessly transformed from a prototype to a product, without sacrificing robustness and scalability. This approach to shorten the time-to-market is our answer for the rapid growth in demand for ESSs. The PLCnext Technology supports development of software using high-level programming languages, which allows the employment of readily available tools. The continuous use of these tools, provides the ability to give long-term quality guarantees to the customer. Furthermore, we present our approach in a way which makes it easy to apply to other industrial systems as well, and can be used anywhere where the quick time to market, scalability and robustness are essential.

Today the various components of the power grid are becoming more intelligent and capable of cooperating in a smart fashion (hence the term "smart grid"). As such, it is important to design the functionalities of the Battery Energy Storage Systems (BESS) in this line of development in such a way that they seamlessly integrate in the grid. These smart systems should include features that automate this system to a degree that very limited user input is required, and is continuously connected and responsive to outside stimuli. In this paper we elaborate on a use case in which the development of these smart functionalities was done for an external manufacturer and retailer. For our particular use case the PLCnext Technology was chosen. For the development of this system

*[1] Nobleo Technology, Eindhoven, The Netherlands
†[2] Phoenix Contact, Zevenaar, The Netherlands

at Nobleo Technology, we make use of the Nobleo Engineering Methodologies. The parties and their interactions are illustrated in Fig. 1.

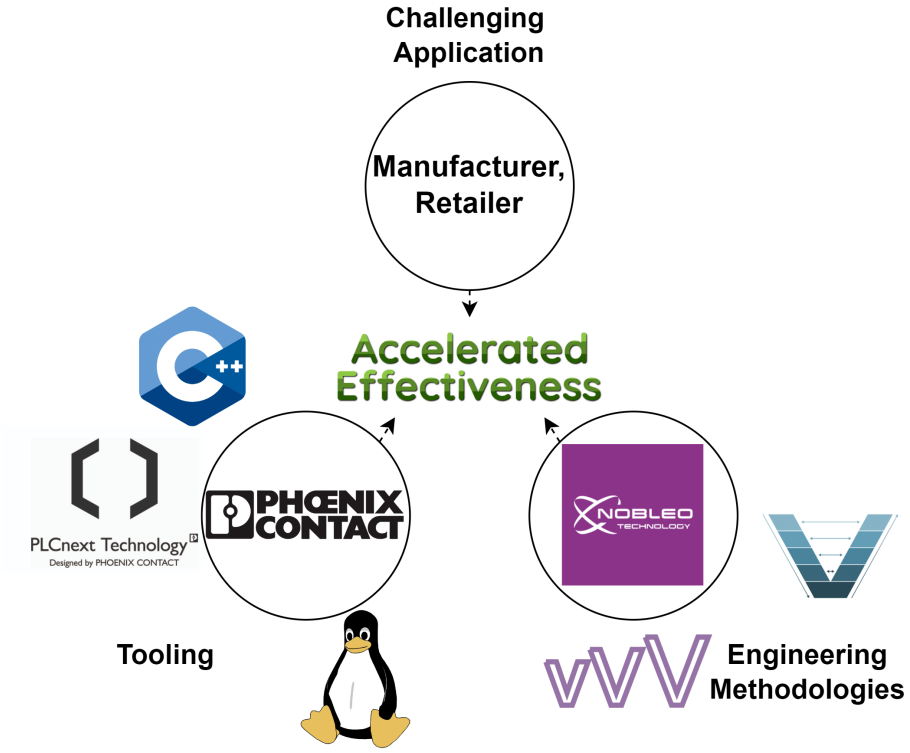

Figure 1: The collaboration of a challenging application, tooling and engineering methodologies leading to accelerated effectiveness

The challenging application is presented by our customers, who manufacture and retail BESS systems. Their wish includes realising the best-in-the-market BESSs with a very short time to market. Users of the BESS only require a limited amount of interactions to get the system running in their desired way. The scope of versatile features that are realised in these systems are illustrated in Fig. 2. The key functionalities are grouped under "internal system control" such as Thermal management, Battery Management and Power control; "user interface" that includes displaying and connectivity and "operation functionalities" containing the operating modes of the system.

Apart from the local activities and functionalities, these systems also connect to the cloud. Using this cloud, the end-users always have insights and control over their system anywhere, as well as enabling support engineers to monitor these systems at all times guaranteeing maximum up-time.

In order to maintain the competitive position in the market, the realisation time of the system is considered very short (+/- 2 months to Functional Model (FuMo) and less than 6 months to field trials). When combining this very short time planning with the wide range of functional wishes, this product asks for an

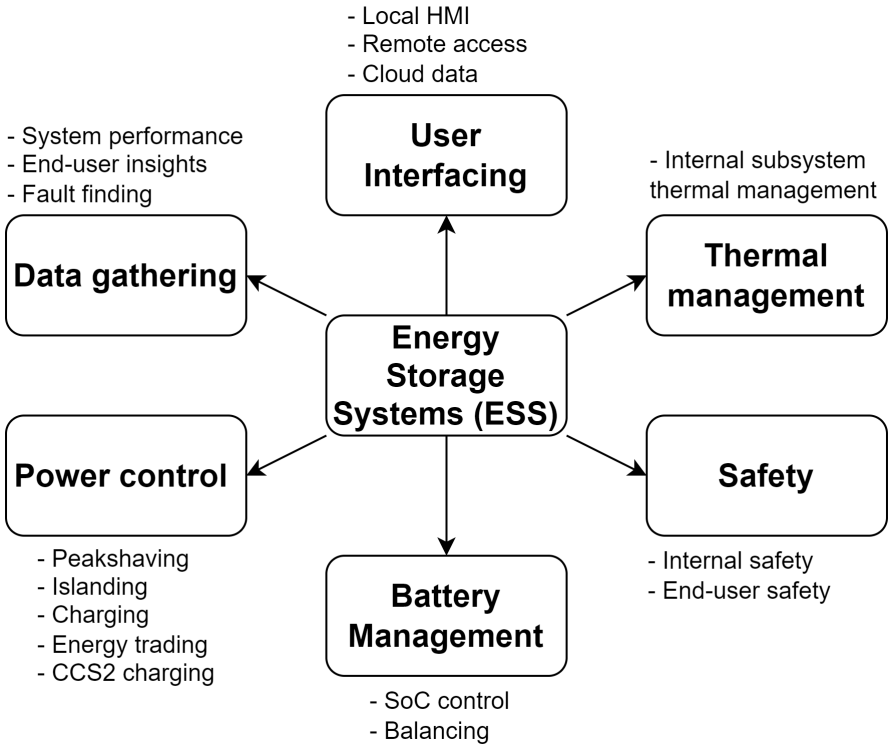

Figure 2: Extent of scope for professional battery Energy Storage Systems.

effective design method (section 3). Secondly, as the market is growing rapidly and the request for features and functionalities are rapidly expanding, there is a need for a flexible and modular embedded core (sections 1 and 2).

Additionally, the designed topology shall allow for (software) field updates without the need of changing the hardware of the system. Thus, the setup that is delivered for the field trials, which is in a prototype state, shall have the ability to be updated to a final product without hardware changes. Using this strategy the ESS are deployed in a relative short timeline maintaining the market advantage without penalising the end-user. This all adds up to a fair challenge in realising this system and requires an effective design strategy.

The rest of this paper is organised as follows. Sections 1 and 2 dive into the current state of the art of controllers; and the benefits and drawbacks of the PLCnext Technology that is selected in this design. This is followed by section 3 that elaborates on the Nobleo Engineering Methods and its advantages. In section 4 the design case of the BESS is discussed. The results of this design case are presented in 5. Finally the experiences and lessons learned are summarised in section 6.

1 Current state of the art in PLC and embedded systems

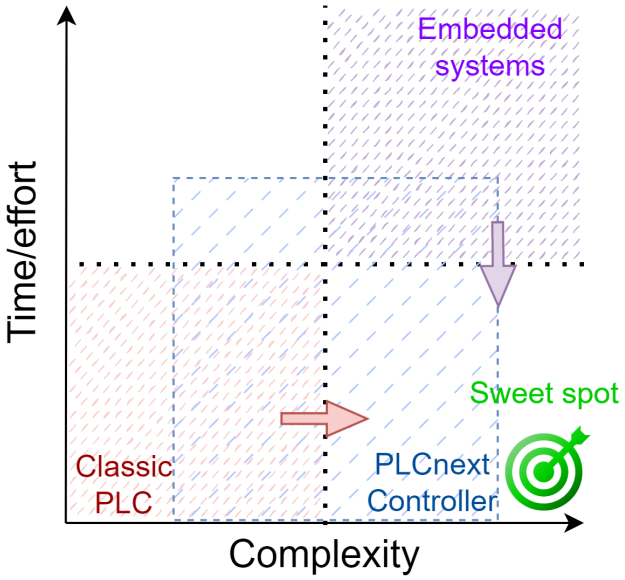

Figure 3: The platform options and their time and effort vs. complexity. The four sections denote the general application areas with respect to their efforts vs. complexity concerns. The target area of bottom-right (high complexity, limited effort) is where our interests lie. The top-right is where Embedded solutions are located, with their high initial effort also clearly implied.

In the current market of industrialisation most trivial system design choices make a trade-off between PLCs and (custom designed) embedded solutions. This trade-off often is based on the time and effort required to build a product and the level of complexity that can be supported by a platform. Ideally one would like to spend the least time and effort to create a desired product with a certain complexity[3]. Fig. 3 visualises this trade-off.

If time to market and modularity are the key factor, often a PLC platform is a logical choice. PLC platforms often offer a large variety of out-of-the-box extensions. The proprietary languages, e.g. ladder and structured text variants, often supported by vendor specific libraries, allow for quick and easy programming of functionalities. Unfortunately, this comes at the cost of the complexity limit of used languages, flexibility of working in parallel in the team and vendor lock-in.

When flexibility and complexity in development are more important than the time and effort, the (custom designed) embedded solutions are the most logical choice. The benefits are the freedom of choosing own processor, in/output methods, protocols, etc. Therefore, this type of platform is a solution tailored to a specific use case. These platforms support a wide selection of programming languages, leaving the many options to the programmers. However, in many

cases a specific hardware is then to be designed. As seen from fig. 3 this platform type takes more time and effort to develop.

Alternatively, a few PLC manufacturers have also started to extend the platforms to support high-level languages (such as python, C++, Matlab) and even provide a full operating system with a POSIX interface. Fig. 4 shows an example of a classic PLC that includes the functionality of translating high-level languages to PLC blocks. This gives the programmer the freedom to program in the language of choice. However, these types of software platforms still contain restrictions in terms of specific libraries that are being supported, as well as limitations on the functionality that can be cast into PLC tasks.

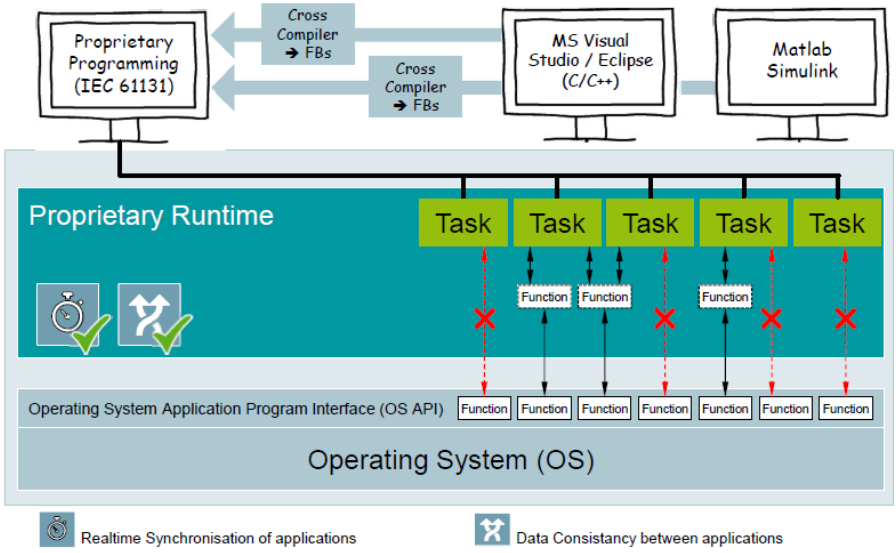

Figure 4: The classic PLC architecture overview. All code is compiled and runs inside a proprietary runtime environment. There is (limited) support for high-level languages that are compiled into a function block which is linked to the proprietary language.

The architecture of a PLC system that includes a Linux kernel is shown in Fig. 5. As shown, a part of the system can be written in the proprietary language, and a part can be created through the language of choice. This works well in platforms where the applications do not need to run in the same runtime (e.g. high-level languages do not need to run real-time), or there is no need to share data between the high-level components and the proprietary components. Hence, these platforms do not natively support functionality synchronisation between the proprietary and high-level languages. If possible, such functionality needs to be developed by the user at the cost of time and extra risks.

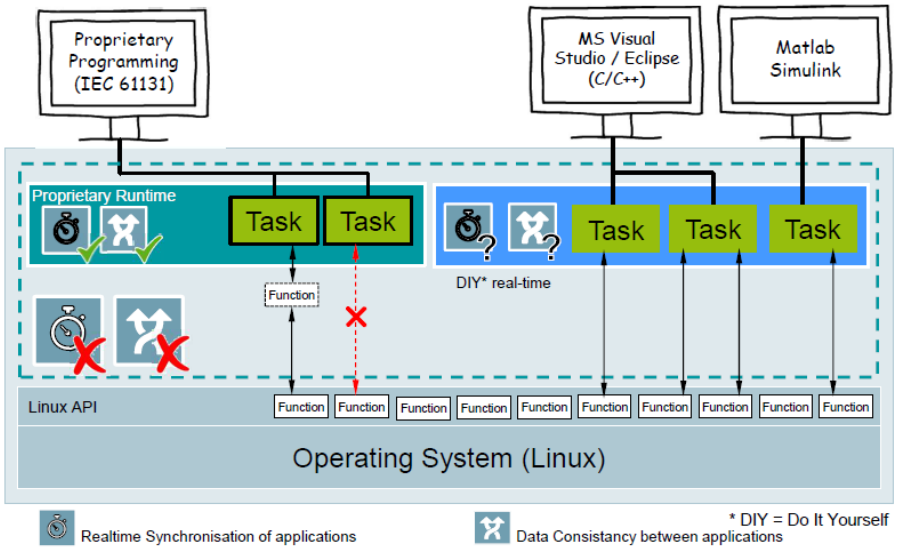

Figure 5: A Linux-based PLC architecture overview. The base PLC program still runs via the proprietary language within its own runtime environment. Code developed and compiled in a high-level language is running separately and therefore not synchronized in the run-time environment, also data is not consistently shared between the separate tasks.

2 The PLCnext Technology

More often than not, a project starts with a very strict time-line, but also has the need of collaboration between team members with varying skills and the development of specific solutions. In these cases a PLC platform with full support for high-level language would offer a good option. In our case however, due to the goal of the product which is power network control, many functionalities require real-time operation.

Phoenix Contact has developed an ecosystem of automation devices called the PLCnext Technology, that is capable of running tasks both described by PLC proprietary means, as well as high-level languages compiled in its Linux environment. A power feature is the ability to combine these two methods of programming, and the aim of this platform is to decrease the time and effort required to build a system, and increase the depth of complexity in the current trade-off (as shown in Fig. 3). A scheduler is created to run both proprietary and externally compiled tasks in real-time; and a memory system, called the Global Data Space (GDS), allows for consistent sharing of data between both. The high-level languages are not limited to the usage of specific subsets and libraries. Instead, the entire tool-chain flow of Linux is available. The architecture of this system is shown in Fig. 6.

This ecosystem supports quick development by having the hardware building blocks and their associated drivers readily available. Yet, when refined and advanced algorithms are required, it also provides the means to program using

a high-level language. Time and project risks are reduced by not having to create an own real-time and data sharing mechanism, and made safe and simple by the fact that consistent cooperation of both refined and simple components is guaranteed. These general features outline our reasoning for selecting the PLCnext Technology to realise the development of our BESSs.

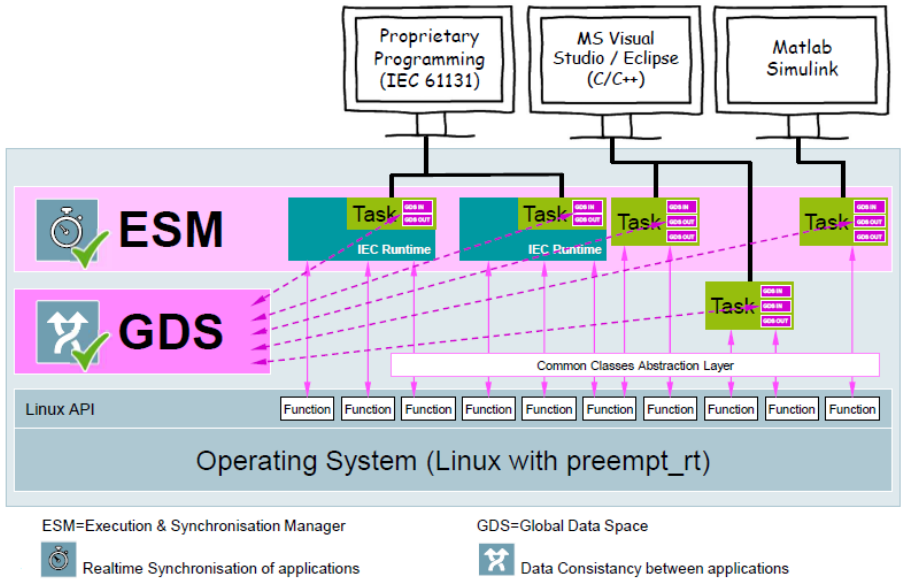

Figure 6: The architectural overview of the PLCnext Runtime System. Notice that this matches the template presented in Fig. 5, and enables real-time synchronisation and a consistent exchange of data of a shared memory space.

3 Nobleo Engineering Methodologies

The V-model is a widely known, and used, system development life-cycle paradigm. The left hand side of the V-model represents activities ranging from the collection and the decomposition of requirements, to their refinement into specifications and the step-wise realisation (i.e., implementation). The right hand side of the V-model represents the integration of components and their validation. The V model also communicates the idea that there should be a correspondence between the activities from the left-hand side and right-hand side. Each refinement from a specification to a solution on any level, is tightly coupled with the reverse activity of integration and directly informs the validation on the same level. This idea is often conveyed by the simple formulae "building the thing right" and "building the right thing".

The V-cycle is a thorough and reliable process to create a product that matches the customer request, but comes at the cost of time. Typically, there is a large time gap between the customer request and the verification with the customer. Nobleo has come up with a spin to the V-cycle with the goal to shorten these development cycles and include the customer early in the development process [4]. The adapted engineering method called the Triple-V

method, which is illustrated in Fig. 7. Instead of developing a full product in one big V-cycle, the triple-V engineering method follows three V-cycles, with increasing complexity, respectively. The focus of the first V-cycle is on the fast creation of a functional model (FuMo), also known as "proof of concept". This gives the customer something tangible to use as a base to refine requirements. The main goal in this cycle is to prove feasibility, create concept designs, and sharpen requirements. Given the refined requirements, a prototype is realised. A prototype generally contains more features (ideally full feature set) to get more data and information on the performance, stability and usability before the production starts. During the evaluation of the prototype, the customer can define what shall be included in the final product. Having learned from the prototype, the final changes and optimisations are implemented. In this final cycle the product is prepared to be manufactured. In the former steps the feasibility, basic functionality and stability were tackled. Therefore, risks are reduced and the final product is ready for production. By making use of the triple-V model, design risks are identified early and the customer is closely involved to assure an effective design. Given the challenging scope and time-line, this engineering method is the best fit to realise the Battery Energy Storage Systems.

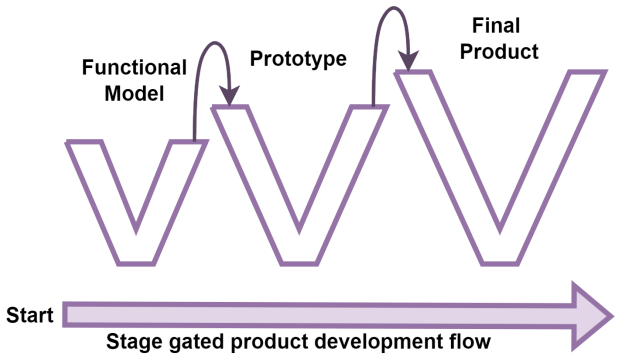

Figure 7: The Nobleo triple-V system engineering methodology which contains three cycles that are extending consecutively from FuMo to Final Product.

4 Design case of the manufacturer

In our use case, the manufacturer and retailer have the vision to create the smartest ESS in the market with a wide range of functionalities on many functional fields as shown in Fig. 2. These functionalities are both local and remote, and combine a mixture of domains. The market calls for various functionalities, which should be configurable from one common hardware platform. In this case we take the example of three distinct product types.

The three product types are visualised in Fig. 8. The first system is used for on- and off-grid AC power and energy purposes such as grid reinforcement. The second system type serves the purpose as a vehicle charger that for example is used for construction projects where no emissions are allowed. The last system type is the energy trading system that solely is connected to the grid.

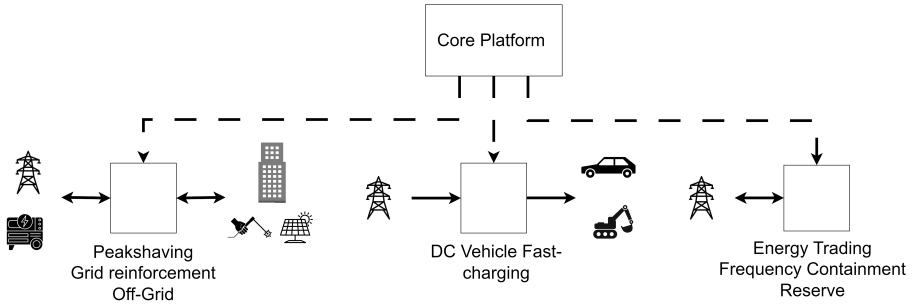

Figure 8: The "core platform" that is developed for the manufacturer is reusable for all system types, leading to a single software platform to maintain.

The commonality between the all of the products is that each is a variant of the same battery energy storage system (BESS). However, each of the three products is adapted to serve a specific market use-case. All three therefore share a a common core in hardware and functionality.

As the majority of functionalities of the ESSs are shared among the products, a system decomposition is created to house all features based on their common denominator. The structural decomposition as created is shown in 9. This decomposition serves as the blueprint for the development of all the products.

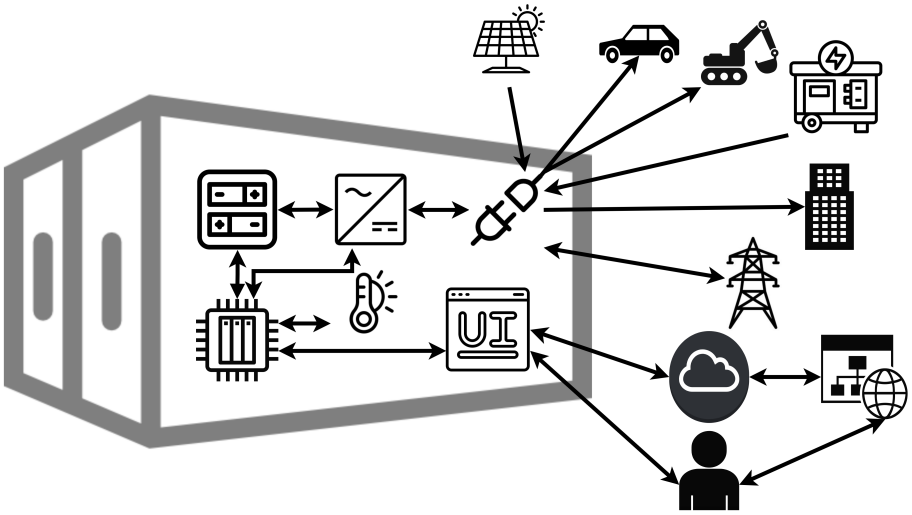

Figure 9: System structural decomposition of the Energy Storage Systems.

The structural decomposition shows that the hardware is abstracted to common architectural parts. The software to operate the ESS system runs in the Energy Management System (EMS). This EMS connects to the majority of subcomponents and houses the majority of system features. As expected, a major part of the software is shared over different system types, therefore, to ensure maintainability a "core platform" is created that runs on all system types. The "core platform" is a mix of the hardware (including the PLCnext Controller,

and other core devices in the EMS) and the codebase. By creating a modular setup, and standardising the "core platform", the complexity of maintenance is decreased, the feature extension is simplified and the addition of new hardware does not negatively impact the high-level system. Moreover, production costs are also decreased by standardising the processes around production and quality control of the core hardware and software.

5 Results

For creating the first version of the system - a basic 3-phase ESS - the customer requirements were very thin. Therefore, a rapid prototyping approach was chosen, which in this case meant developing a FuMo with a small team of 3 engineers in a short time-span, such that at a later stage the project could be tailored to the customer's wishes. The FuMo was built within two months (Fig. 10) and served as the first verification towards the feasibility of the project and the solutions therein. Various solutions were chosen to realise this FuMo, for instance on the software-side the PLC language (Structured Text) was employed more extensively, with the aim of replacing the components with more classical and better expressive high-level languages in later stages. While using this language brings limitations to the complexity of the functionalities, it enabled the possibility to quickly verifying the system integration. The limitations sometimes forced non-elegant solutions and undermined the creation of a modular design, which however was tolerated due to the nature of the project. The additional incurred effort was offset by the fact that this step gave more insights into the risks and helped the customer refine their requirements early. This resulted in a substantial percentage of the codebase being written in the proprietary language, compared to the C++ language. This fact is demonstrated in Table 1.

After the completion of the FuMo and verifying the minimal viable product (base functionality, happy flow) with the customer, the next set of system requirements was defined. These requirements served as the input for the creation of a prototype. As the maturity and complexity of the system functionalities increased, the need for high-level programming languages increased. Secondly, the team capacity increased to realise the requested functionalities. Due to both

Codebase	C++ lines of Code	Structured Text lines of Code	Total	C++ Percentage
FuMo (Sep. 2022)	7,047	2,523	9,570	74%
Proto (Dec. 2022)	12,519	2,589	15,108	83%
Pre-production (April 2023)	35,130	4,304	39,434	88%
Current Product (Jan. 2024)	56,387	5,639	63,827	91%

Table 1: Proportion of high-Level language implementation within the codebase over time

these factors and also employment of tooling (version control, CI/CD) which was available for high-level language development, the ability to perform tasks in parallel by the team members also increased over time.

The code that was developed in the existing proprietary language served as a base, and remained there to be used in the hardware interface. This served two purposes. Firstly, the full system was already proven functional. By replacing low-level complexity items with full-fledged high-level components, the base functionality remained equal. Second, this gave the freedom to start replacing the software components in phases without the need of making hardware changes. Depending on the complexity requirement for each functionality the replacement was scheduled and realised whilst the system remained fully operational for testing and validation. It took only three months (Fig. 10) to build a prototype that was available for customer field trials. As earlier discussed in Table 1, we can see that at this stage the share of the proprietary language in the codebase decreased by 35%.

The first prototypes served as field trial systems used by end-users, and useful data was gathered from these systems. The manufacturer was able to test, verify and validate these prototype systems against the end-user expectations before the development into a full-fledged product was resumed. In the prototypes, the majority of the features of the systems were captured, however, mainly these included the sunny-day scenarios. The software included a limited capability to automatically handle errors and a limited amount of code was covered by automated software-in-the-loop tests.

By evaluating the performance of the prototypes together with the manufacturer, the project continued into the product development phase. Not only shall the product function in the happy flow, additionally it should also be able to handle the warnings and error cases. In order to extend the stability of the product the amount of automatic software tests were extended, and code check mechanism were implemented. By doing so, faults could be caught before even having occurred in the field at the customer side.

These tests and checks did require the code to be a high-level language (HLL), as for these type of languages there are plenty of tools available. Therefore, it is important to have a codebase with a high proportion of C++ code in this project. As discussed in Table 1 the amount of proprietary language lines of code were again decreased by another 30%.

After the product development phase, the project was transferred to the manufacturer to start the normal Life Cycle Management. From there, the current codebase of the project shows that the proportion of the C++ code has increased even further.

From Table 1 it can be deduced that the amount of proprietary lines of code did extend throughout the process compared to the start. As we expected that during the process the proprietary language was replaced by C++ code as much as possible, this can be seen as a strange phenomenon. However, the main link between C++ blocks and hardware and C++ blocks is done via the GDS. As previously discussed, the GDS is a mechanism in the PLCnext runtime system to consistently and efficiently exchange data between components. This is done by the PLCnext Technology's proprietary framework. The specifications of the GDS ports and their connections are translated into files, and thus count as "Structured Text lines of Code". Therefore, if the complexity of the software increases, the amount of links between components increases which leads to an

increase in the amount of lines of proprietary code.

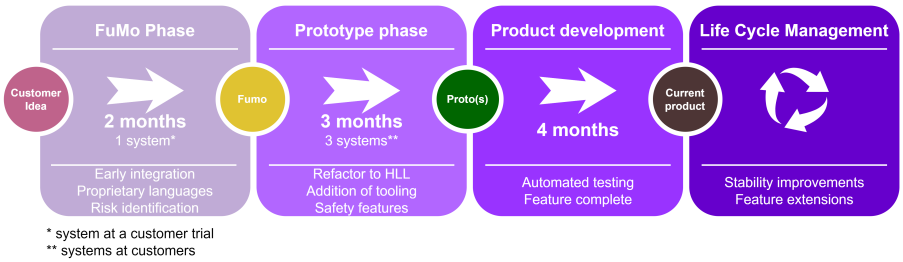

Figure 10: Timeline of the project development steps and their outcomes showing the impact of the Nobleo design methodologies to develop a complex product in a very short time-span.

6 Conclusion

By having realised the first MVP ESS in two months and the prototypes three months later, we have shown the positive impact of the PLCnext Technology and effectiveness of Nobleo design methodologies in the design of Energy Storage Systems (ESSs). Moreover, the additional choice of the PLCnext Technology, tools and the Nobleo triple-V model facilitated the seamless move from prototyping to production. The ability to develop software using high-level programming languages (in this case C++) allowed us to use a plethora of off-the-shelf tools to help in validation and quality improvement of our software base. Moreover, the continuous use of these tools, enables us to provide long-term support and quality assurances to the customer.

However, there were also lessons learned regarding how this process can be improved in the future. Throughout our development we learned that building structures to gather and analyse diagnostics data early on, provides a lot of value to developers and speeds up the arduous task of debugging and diagnosing issues in products. Additionally, building and / or integrating software support tools, such as automated build tools and static analysers, remote monitoring solutions, which initially were not seen as essential proved to make a significant difference in the long run.

We believe that these lessons and experiences are not exclusive to the development of ESSs, and are readily translatable to other types of industrial systems, and can be used in any context where quick time to market, scalability and robustness are essential.

References

[1] M. Storm van Leeuwen, P. Oortman, "Bancheanalyse 2022-2023", Energy Storage NL, 2023

[2] Sverre Alvik et al. , "Energy Transition Outlook 2023", DNV-Group, 2023

[3] "Heeft de PLC nog een toekomst?"(whitepaper), Phoenix Contact, 2021

[4] Getting grip on your development process: Our Way Of Working. (2023). Nobleo Technology. https://nobleo-technology.nl/projects/getting-grip-development-process/