Production Engineering (2022) 16:847–856
https://doi.org/10.1007/s11740-022-01133-y

PRODUCTION MANAGEMENT

Automation software architectures in automated production systems: an industrial case study in the packaging machine industry

Eva-Maria Neumann[1] · Birgit Vogel-Heuser[1,2,3] · Juliane Fischer[1] · Sebastian Diehm[4] · Michael Schwarz[4] · Tobias Englert[4]

Received: 11 August 2021 / Accepted: 13 April 2022 / Published online: 16 May 2022

Abstract
In the era of Industry 4.0, advances in production engineering are driven by modern machines and equipment, whose evolution depends primarily on software nowadays. These machines are combined in automated Production Systems (aPS), whose software is characterized by high complexity, long lifetimes, and strong coupling of mechatronic disciplines. The development of modular, flexible software architectures that adapt to company- and process-specific boundary conditions is an essential prerequisite for companies to compete globally. While there are many approaches in computer science, a clear definition of control software architecture in aPS and systematic approaches to analyze company-specific software architectures and the underlying design decisions are still missing. This gap is addressed by defining control software architecture in aPS, including architectural views to address the heterogeneity of influencing factors on control software. To enable a systematic architecture analysis, templates are defined for visualizing design decisions to derive concrete recommendations to support practitioners in improving software. An in-depth interview study in three renowned companies from packaging machinery confirmed the benefit of the proposed architecture analysis to systematically identify optimization potentials and concrete starting points for the implementation.

Keywords Automation software architecture · IEC 61131-3 · Design decisions · Automated Production Systems

1 Introduction and motivation

The current technological advances in production engineering require highly-flexible, evolvable machines and equipment combined to so-called automated Production Systems (aPS) [1]. Nowadays, the evolution of aPS is predominantly realized via software and thus places high demands on control software architecture. While there are numerous software architecture (SWA) analysis approaches in the embedded systems area, this field has been hardly researched for control software architecture in aPS (CSWA). The hard real-time requirements, lifecycles of several decades, and code change during runtime of the operating machine [2] hamper the development of universal guidelines for high-quality architecture. However, a

The Institute of Automation and Information Systems thanks Harro Höfliger Verpackungsmaschinen GmbH, SIG Combibloc Systems GmbH, and SOMIC Verpackungsmaschinen GmbH & Co. KG for taking part in the interview study and for providing insights into their software and processes.

✉ Eva-Maria Neumann
eva-maria.neumann@tum.de

Birgit Vogel-Heuser
vogel-heuser@tum.de

Juliane Fischer
juliane.fischer@tum.de

Sebastian Diehm
sebastian.diehm@se.com

Michael Schwarz
michael.schwarz@se.com

Tobias Englert
tobias.englert@se.com

[1] Institute of Automation and Information Systems, TUM School of Engineering and Design, Technical University of Munich, Garching, Germany

[2] MDSI, Munich, Germany

[3] MIRMI, Munich, Germany

[4] Schneider Electric Automation GmbH, Marktheidenfeld, Germany

cleanly modularized, evolvable, and maintainable CSWA is the key to implementing future-oriented technologies and, thus, staying competitive.

A major challenge in improving CSWA often lies in recognizing influences between design decisions and identifying concrete starting points for optimization. This paper, therefore, presents an approach to define CSWA for aPS based on previous analyses of industrial use cases, including five architectural views for considering influencing factors on CSWA. Using an in-depth industrial interview study with renowned control software experts of three packaging machinery companies, the suitability of CSWA types for different company-specific boundary conditions and production processes is investigated. Templates for documenting architectural design decisions are introduced to visualize the connections, drivers, and consequences to derive recommendations for improving the architecture. The benefits of the architecture analysis and the derived recommendations are proven in follow-up meetings with the interviewed experts.

In the following, first, the state of the art of analyzing SWA is introduced (Sect. 2) to derive a definition for CSWA (Sect. 3). Section 4 introduces morphological boxes to structure influencing factors on CSWA. Section 5 defines templates for documenting design decisions and formulating recommendations for action used for the interview study described in Sect. 6. The paper closes with an outlook in Sect. 7.

2 Related work on SWA of production systems

The following subsections outline the state of the art of control software in aPS and analyzing SWA.

2.1 Boundary conditions of CSWA in aPS

aPS are mechatronic systems of high complexity [2], usually programmed using Programmable Logic Controllers (PLCs) characterized by cyclic program execution. PLCs are mainly programmed according to the IEC 61131-3, defining two textual and three graphical languages and structural elements, i.e., so-called Program Organization Units (POUs) as reusable software units. Since 2013, the object-oriented programming paradigm is officially integrated into IEC 61131-3 (OO-IEC [3]) as object-oriented language elements analogous to high-level language programming. However, it is currently barely used in PLC programming, e.g., because the software is often maintained by technicians without a software background, who can handle a procedural programming style more easily [4].

2.2 Related work on analyzing SWA and quality

In computer science, one of the most cited definitions of SWA is "all major facets of a software system, including its structural elements—components […], connectors […], and configurations" [5]. These fundamental elements of SWA are confirmed in further established definitions [6, 7]. SWA is determined by design decisions, which need to consider the system's non-functional properties [5]. Desirable SWAs solving recurring issues can be formulated as reusable design patterns to support software developers facing similar issues [5, 8]. The architectural design directly affects the software's quality attributes, including performance, development cost, or maintainability [9].

Previous industrial analyses [10] revealed the challenges of defining well-structured CSWAs in aPS since PLC software must cope with boundary conditions that strongly differ from the ones in embedded systems, including platform constraints such as limited cycle times or software changes during operation of the running system [2, 4].

Static code analysis [11] is beneficial for measuring SWA quality attributes, e.g., using software metrics [12]. Vogel-Heuser et al. [10] identified a typical five-level module architecture in PLC software, which complies with the ISA-88 levels and comprises modules of different granularity, Maga et al. [13] controlling individual actuators such as cylinders or drives (basic modules) up to controlling the behavior of whole machines (facility modules) or plants (plant modules). However, an exact definition for CSWA in aPS and systematic approaches for evaluating its strengths and weaknesses to provide concrete recommendations for optimization are still missing.

3 Definition of CSWA in aPS

The basic elements of SWA definitions in computer science can be also be found in CSWA. *Components* can be POUs (individual ones or groups arranged to modules), actions, variables, or OO-IEC elements (e.g., properties or methods), which are *connected* by data exchange (calls or reading/writing variables) or by structural connections resulting from OO-IEC, i.e., inheritance or interfaces. However, the core characteristics of CSWA, such as the need for hard-real time and software changes during operation, are not considered in definitions from computer science [2]. To address this gap, representative control software projects and architectural guidelines of three companies with different boundary conditions are compared (cf. Table 1), i.e., a packaging machine manufacturer

Table 1 Overview of use cases to derive the architecture definition

Use case	Industrial sector	aPS type
Company A	Packaging machinery	Series machine manufacturing
Company B	Automotive	Plant manufacturing
Company C	Woodworking machinery	Plant manufacturing

(A) with mature CSWA and quality management as proven in a preceding questionnaire study [10], a plant manufacturer (B) from automotive applying precise programming guidelines to ensure high software quality, and a plant manufacturer from the field of wood industry (C).

The results show that CSWA is strongly influenced by the particularities of aPS elaborated and confirmed in preliminary work [2] and is mainly determined by design decisions in the following architectural views, i.e., specific subsets of the system's structural elements [14].

Hardware influences on hierarchy and modularization The significant hardware complexity in aPS and the long lifetime require mature CSWAs to enable evolvability during the system's operation via software changes during runtime [2]. Depending on the aPS size and motion tasks, different automation hardware architectures are required (e.g., usage of multiple PLCs or sophisticated drive technology), which also strongly influence the CSWA.

Reuse strategies CSWA is affected by the applied reuse strategy, ranging from systematic reuse of quality-tested library POUs or software templates to unplanned reuse via Copy, Paste & Modify, which is still predominant in CSWA development, usually leading to uncontrolled software growth and drawbacks regarding maintainability.

Extra-functional tasks Extra-functional tasks such as exception handling, connection to the human–machine interface (HMI) or operation mode switching account for up to 75% of an control software project [15]. These tasks are usually modularized differently from the functional software parts but are also closely coupled with them, making variability management for CSWA a major challenge [16].

Programming paradigm and software development Software changes during operation are often carried out by technicians with little programming background making it difficult to apply object-oriented programming paradigms that are standard in computer science [2]. In addition, certain OO-IEC constellations can cause runtime issues and thus conflicts with the hard real-time requirements of aPS [17].

Company-specific boundary conditions This is not an architectural view per se since company characteristics cannot be mapped directly to the software. However, company-specific boundary conditions, e.g., the workflow of interdisciplinary cooperation [4], strongly influence CSWA.

Summarizing, the consensus of SWA definitions from computer science is not sufficient to define CSWA. Thus, to understand and optimize CSWA, the definition is enlarged by the architectural views as introduced above.

4 Morphological boxes for architectural influencing factors

Since the analysis and comparison of architectural design decisions is a multidimensional problem, morphological boxes are introduced to describe different architectural views. The morphological boxes are derived from previous industrial case studies [10] and substantiated by consultations with experts from academia and industrial automation.

While there is a broad consensus in high-level language software on desirable design principles leading to high quality (cf., e.g., [18]), the definition of universal best practices for CSWA in aPS is challenging due to company-specific differences in the understanding of software quality and various stakeholders with different background working on the software (Table 2). In plant and special-purpose machinery, e.g., technicians with little programming skills often need to change software under time pressure during commissioning, thereby potentially impairing CSWA in case it is not intuitively understandable. Experience of industrial code analyses shows, e.g., that larger companies, especially when operating at multiple locations, are more urged to cleanly modularize and document the software since an exchange "on-demand" during development is hardly possible, e.g., to compensate for difficult-to-understand code fragments. Depending on the industry, there can be specific standards (e.g., OMAC for packaging machines) or legal requirements (e.g., in the medical sector) that require or support the maintenance of a high-quality CSWA (cf. Table 2). Safety-related standards such as the *Good Automated Manufacturing Practice 5 (GAMP 5)* in the pharmaceutical and food industry enable risk assessment throughout the system's lifecycle to ensure product quality and safety. Certification according to GAMP requires the fulfilment of design specifications by the CSWA and the integration of validation procedures into the system's development workflow.

Regarding *hierarchy and modularization*, experience from previous case studies [10] shows that machines, which are expendable by well-defined reusable stations to address different customer needs, also require a well-modularized CSWA (cf. Table 3). Unlike direct data exchange via POU interfaces and calls, indirect data exchange via global variables often inhibits reusability. A hierarchical CSWA structure is generally rated as beneficial. Structuring the CSWA towards the physical layout of the aPS enhances the system's understandability and thus facilitates its evolution and maintenance [10].

Table 2 Morphological Box to classify company-specific boundary conditions

Criterion	*Options*						
Company size	small (<250)		medium (251 - 1500)		large (> 1500)		
Locations	one central location		multiple locations in Germany (++)		multiple international locations (++)		
Number of programmers working on a software project	1 - 10	11 - 20	21 - 50 (++)		51 - 100 (++)		
Distribution of programmers working on a software project	work centrally at one location in Germany (on call possible)		work distributed over several locations in Germany (++)		work distributed over several international locations (++)		
aPS type	Series machine manufacturing (++)		Special purpose machine manufacturing (−)		plant manufacturing (−)		
Industrial sector	Food industry machinery (+)	Process Engineering (−)	Wood working machines	Packaging machines (+)	Pharma / Medical Technology (+)	…	
Company-specific programming guidelines	yes (++)		partly (+)		no (−)		

Legend: *Positive green (+/++), neutral (white), or challenging (orange (-/--)) influencing factors*

Copy, Paste & Modify is rated to have a strong negative effect on the CSWA (cf. Table 4) due to the reasons derived in Sect. 3. On the other hand, systematically reusing control software, e.g., using templates, libraries, or code generation, is expected to enhance CSWA quality.

5 Templates for analyzing architectural design decisions and recommendations of action

Documenting and, thus, understanding industrial CSWA design decisions requires a comprehensible form of presentation that provides a quick overview of the design decision itself, the drivers leading to it, its consequences, and where to find it in the software. Therefore, the following four-part template is proposed (see Fig. 1).

Analyzing the design decisions and their consequences allows a clear understanding of the strengths and weaknesses of the existing CSWA to identify starting points for CSWA improvement and to assess influences between planned adaptations and the design decisions already made. Therefore, understandable recommendations for action are required, which cover the following aspects that are based on [19] and enlarged by precise categories for the individual aspects (Table 5):

6 Interview study to analyze architectural design decisions in the field of packaging machinery

The applicability of the architecture analysis using the templates of Sect. 5 and the assumptions on influencing factors on CSWA (cf. morphological boxes in Sect. 4) are analyzed by conducting expert interviews in three aPS manufacturing companies.

6.1 Comparability of companies

To ensure the comparability of the results, companies from the same industrial sector using PLC platforms of the same supplier are analyzed. Thus, the companies are confronted with similar challenges (e.g., similar complexity of motion tasks) but also have similar resources to cope with these challenges, such as solutions offered by the platform supplier or domain-specific standards. On the other hand, to investigate the impact of different company-specific boundary conditions and production processes, the

Table 3 Morphological Box to classify hierarchy and modularization

Criterion	Options					
Size of the machine	Differs, but expandable with reusable stations (++)		Stations are reused via Copy, Paste & Modify (−−)		uniform size (+)	
applied progamming languages	Structured Text (ST)	Instruction List (IL)	Sequential Function Chart (SFC)		Ladder Diagram (LD)	Function Block Diagram (FBD)
Data exchange direct via interfaces / POU calls	yes (++)		partly (+)		no (−)	
Indirect data exchange, e.g. via global variables	yes (−)		partly		no (+)	
Standardization of modules	Plant modules (++)	Facility modules (++)	Application modules (++)	Basic Modules (++)	Atomic Basic Modules (++)	not standardized (−−)
Structure of the software	hierarchical (++)				flat	
Available module hierarchy levels	Plant modules	Facility modules	Application modules (++)	Basic Modules (++)	Atomic Basic Modules	
Modularized accoring to physical machine layout	yes (++)		partly (++)		no	

Legend: *Positive green (+/++), neutral (white), or challenging (orange (-/--)) influencing factors*

Table 4 Morphological Box to classify reuse strategies

Criterion	Options				
Universal Modules	yes (++)				no
Templates ("150% Projects")	yes (++)				no
Usage of Libraries	customer-specific (++)	machine-specific (++)	supplier libraries (++)	Company-wide standard libraries (++)	are not used
Usage of Code Generation	standardly (++)	often (++)		rarely (+)	never
Copy, Paste and Modify	standardly (−−)		partly (−)		no (++)

Legend: *Positive green (+/++), neutral (white), or challenging (orange (-/--)) influencing factors*

interviewed companies show significant differences in the number of employees, locations, and requirements for the respective machines (cf. Table 1). Food and packaging machinery is one of Germany's most important industrial sectors in machine and plant manufacturing and is a highly heterogeneous industry [20]. Therefore, it is ideal for analyzing CSWA design decisions under different boundary conditions. This leads to selecting the following three companies (cf. Table 1).

Table 5 Template for formulating recommendations for actions

Category	Specification
Recommendation (Selection of one option)	Type 1: **Addition/deletion/modification** of components, connectors or configuration
	Type 2: Appliance of an **architecture analysis method** to monitor/understand the architecture
Details (Both options need to be specified)	**Reason**, why the current solution is not optimal
	Explanation how the recommendation solves this problem
Architectural View (AV)	*AV1:* Main View (Hierarchy and Modularization, Reuse, (Extra-)functional Tasks, Programming Paradigm)
	AV2: Optional specification of affected individual aspects of the respective AV1
(Extra-) Functional Task	**Functional**: Application logic (on plant, facility, application, basic, atomic basic level)
	Extra-functional: Operation Mode Switch, Fault Handling, Linkage to HMI, Operating Data Collection, Hardware Control, Other/Further
Non-functional property	Non-functional attributes of the architecture affected by the recommendation Classification according to ISO 25010
Scope	*S1:* Whole software structure (project level)
	S2: Modules, POU libraries, POU constellations for extra-functional tasks
	S3: Individual POUs, library elements
	S4: Functionally related code sections within POUs
	S5: Individual Operators/Operands

- *Company D* with approximately 400 employees operating globally with one central location producing machines for the final packaging for pre-packaged products (food and beverage)
- *Company E* with more than 1400 employees operating in around 40 countries producing machines for packaging medical and pharmaceutical products
- *Company F* with approximately 5500 employees operating in around 40 countries with strict requirements on the aseptic packaging of food and beverage products

All companies apply company-internal programming guidelines to enhance and maintain their software quality.

6.2 Interview conduction and results

To evaluate the relevance of the architectural views on CSWA (cf. Sect. 3), questions on all views are formulated in cooperation with experienced practitioners from industrial automation. These questions are discussed in each company within 3-h interviews with eight software experts from the companies, including PLC programmers, HMI experts, and project managers, to cover different perspectives on CSWA. The interviews are conducted by a mixed team of two academic researchers and three senior engineers from the platform provider to classify the CSWA both in the context of preceding research and in the current state of technical practice. The identified design decisions and derived recommendations are prepared using the templates introduced in Sect. 5 and discussed in follow-up meetings with the interview participants to obtain feedback on the results, prevent misunderstandings, and clarify open questions.

In the following, the analysis results in the views *hierarchy and modularization* and *reuse strategies* are introduced.

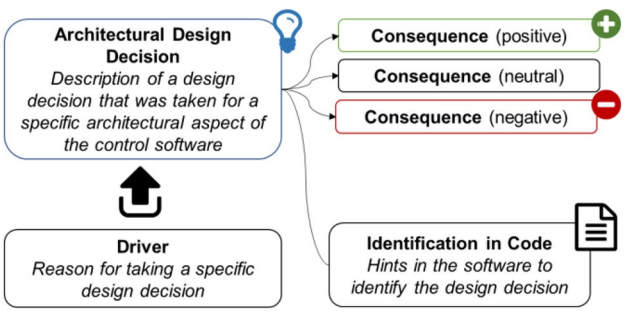

Fig. 1 Template for documenting architectural design decisions

6.2.1 Hierarchy and modularization

Company D follows a strict modularization approach oriented towards the functional structure, which serves as the common basis between the involved disciplines and is, therefore, also reflected in the physical layout of the machine. The central element is the so-called *Functional Unit (FU)*, a clearly defined sub-function of the machine controlled by a corresponding POU. The function-oriented modularization makes the structure intuitively understandable across disciplines. However, there are FUs without direct hardware representation, e.g., extra-functional software parts used across modules, which hamper the maintainability of the CSWA.

The software hierarchy in Company E is oriented towards the physical machine layout starting from the *MainMachine*, calling the underlying modules controlling autarch machine

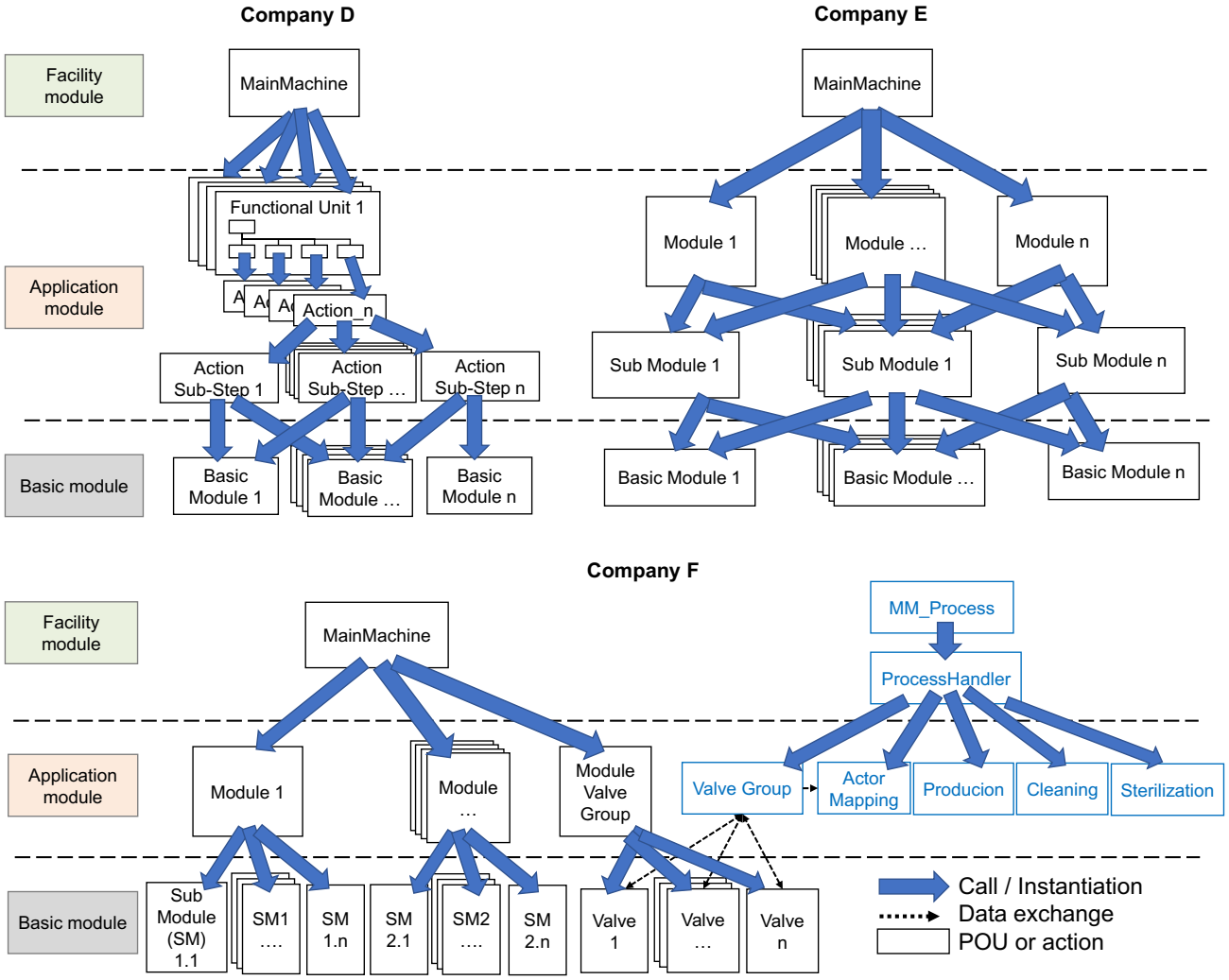

Fig. 2 Call Tree Structure in the three investigated companies (visualization oriented towards [9])

parts, e.g., for feeding parts to be packed together. The modules call the required stations, i.e., process steps that belong together. Underneath the stations, basic modules to control individual actuators such as cylinders are called.

To meet the stringent aseptic requirements for packaging food and beverage products, Company F chooses a CSWA in the form of two parallel tree structures—one to control the machine behavior and one to control the production cycle consisting of production, cleaning, and sterilization to ensure continuous compliance with hygiene requirements (Fig. 2, bottom right). The software controlling the machine behavior is oriented toward its physical layout. The *ProcessHandler* ensures compliance with the production cycle and allows only specific process steps. Depending on the current production cycle phase, the *ActorMapping* specifies the possible output values of the valves in the *ValveGroup*.

Comparing the software hierarchies shows that all companies decided to structure the software in a hierarchical tree-like pattern reflecting the physical machine layout. However, the different boundary conditions cause the trees to take different shapes. The interviewed experts confirmed that although a hardware-oriented modularization approach increases the comprehensibility, it is usually not applicable for the whole software, since, e.g., extra-functional tasks have no direct representation in the hardware and thus do not fit into the structure, e.g., the production cycle in Company F.

6.2.2 Reuse strategies

The companies apply different strategies for reusing control software, each tailored to their respective boundary conditions and requirements. Company D uses a universal

template, i.e., a master project containing all theoretically available FUs and sub-modules. All unnecessary parts are deleted to create a concrete machine project. A prerequisite for this design decision are the small team sizes, which enable work on-call and thus keep the probability of errors low due to good communication. However, the universal template is challenging to understand for new employees due to its large size and is laborious to maintain. In addition, the reuse method leads to a high proportion of dead code since empty FBs must remain after reducing the master project to enable compilability without modifying calling POUs. New variants are created using Copy, Paste and Modify of existing FUs. OO-IEC, which would enable new variants using inheritance, is not widely accepted by older employees. However, the programmers of Company D criticized an increased maintenance effort and inefficiency due to double implementation.

Company F decided on a reuse concept based on mature module libraries combined with OO-IEC. In addition to supplier libraries, company-internal libraries are applied, e.g., for error handling. For Company F, a universal template as in Company D would not be reasonable due to the high complexity of the CSWA and the much larger teams. To avoid Copy, Paste, and Modify and thus, the long-term quality and maintainability issues, FBs that are potentially suitable for reuse are stored in a separate folder to be standardized for later inclusion in the libraries.

Company E develops most of the software using code generation based on design decisions taken in mechanical and electrical engineering. A new project is generated for each machine based on reusable templates (cf. Fig. 1). Some (sub-)modules (e.g., for cartoning) are used in many different machines, resulting in a high level of reuse. One advantage is the cross-disciplinary consistency in engineering. However, during the generation process, POUs of very small granularity are sometimes created, hampering the software's understandability.

Regarding reuse of control software, it can be concluded that the applied reuse strategies are tailored to the company-specific requirements and challenges and would not fit the respective other companies without drawbacks or necessary adaptions.

6.3 Documentation of results and recommendations for action

The results of the interviews are prepared using the template (cf. Fig. 1) to derive recommendations for action, which, together with the design decisions, are evaluated with the interview partners. The approach is demonstrated using the design decision of Company D to reuse software based on a universal template.

Applying the template to the design decision (cf. Fig. 3) reveals that many advantages accompany the reuse strategy

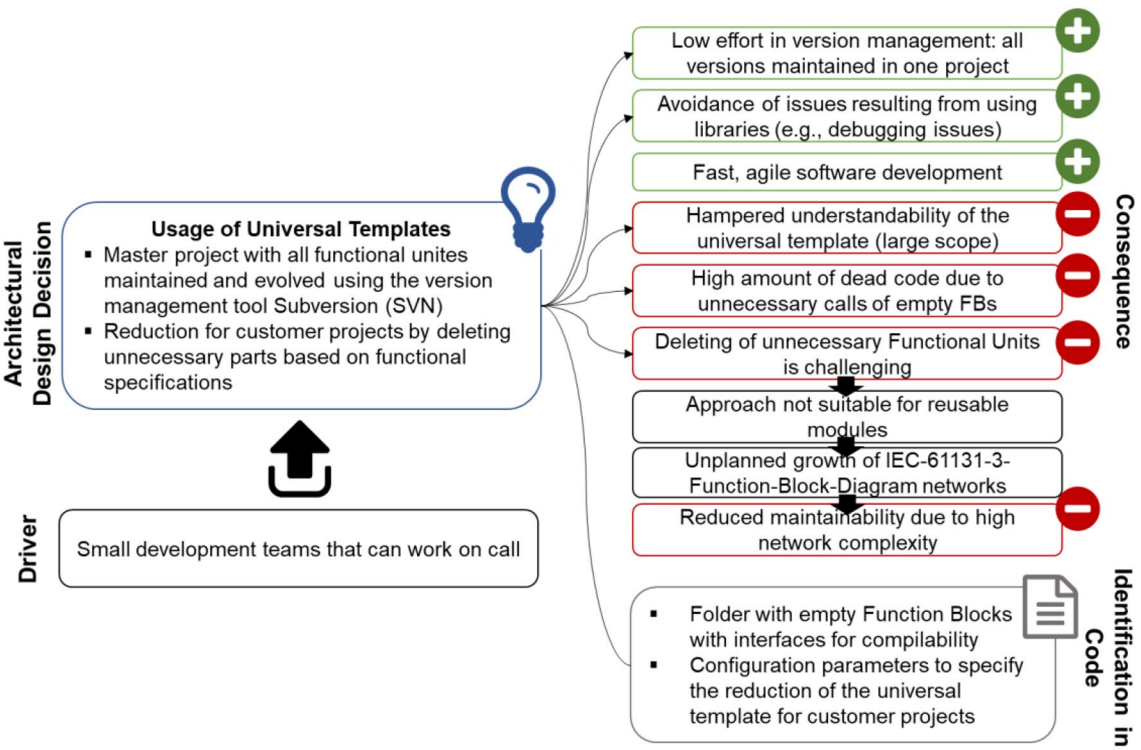

Fig. 3 Structured visualization of the design decision of Company D to use universal modules using the proposed template

Table 6 Recommendation for action to address the hampered understandability of the universal template (Company D)

Category	Specification
Recommendation	Type 2: Visualization of the structure and functional distribution of the master project
Details	**Reason**: Large scope of master project and therefore difficult to understand
	Explanation: Visualization facilitates orientation in the project and prevents errors
Architectural View (AV)	*AV1:* *1.* Hierarchy and Modularization *AV2:* 1. Structure of the Software 2. Data Exchange 3. Available module hierarchy levels
(Extra-) Functional Task	Both functional and extra-functional tasks
Non-functional property	Maintainability (including modularity, reusability and analyzability)
Scope	*S1:* Whole software structure of the master project

as it is tailored to the company-specific boundary conditions. However, it also reveals the disadvantages outlined in Sect. 6.2. These negative consequences are then addressed by concrete recommendations applying the table schema (cf. Table 4), in the following exemplarily demonstrated for the disadvantage "Hampered understandability of the universal template" (cf. Table 5), which can be enhanced by visualizing the structure and functional distribution analogous to Fig. 1 (see Table 6):

All participants confirmed that visualizing the CSWA analogous to Fig. 1 enhances understandability and supports the familiarization of new employees with the software architecture. The template-based representation of design decisions proved to be a helpful means of making software architecture and the connections between different decisions comprehensible. In particular, the analysis of CSWA from different views provides significant benefits since only the combination of information from different views and their interrelationships allows the systematic, target-oriented planning of optimizations of the existing software. Moreover, the analysis of design decisions in different views enables the systematic derivation of recommendations to improve CSWA. This was strengthened particularly during the follow-up discussion in Company D: Some of the derived recommendations for action have already been identified by the company itself, and implementation is already in progress. The experts considered the table-like format very helpful, and the template's granularity is sufficient to understand and implement the recommendations.

7 Conclusion and outlook

This paper proposes a definition of CSWA by introducing architectural views that consider the particularities of aPS affecting control software [2], including the complexity of hardware layout and motion tasks, the need to adapt to changes during operation, and the challenges of implementing extra-functional tasks. The impact of the architectural views is examined with experts from academia and industrial automation based on morphological boxes. To analyze and enhance industrial CSWA, templates are defined to document and analyze architectural design decisions and their impact and formulate concrete recommendations for action to enhance an existing CSWA. The benefit of the templates is demonstrated by an industrial interview study in three packaging manufacturing companies, and the identified design decisions in different architectural views are compared and discussed.

The positive feedback from the experts in the interview study shows the great potential of a systematic architecture analysis to optimize existing software and enable the implementation of pioneering technologies in production engineering. Therefore, future research will analyze how such an architecture analysis can be integrated into the industrial software development workflow. Especially for small companies, it is financially and capacity-wise unrealistic to set up separate departments for systematic quality management, so their empowerment to optimize CSWA with given resources efficiently will be the focus of future research. Currently, the analysis is done manually, but there are already considerations regarding which aspects of the analysis can be automated to facilitate the implementation in industrial practice. This could, e.g., identify violations of existing architecture specifications. In addition, the combination of different views in the architecture assessment and the resulting knowledge about influences between design decisions can support the development of new software projects in the design phase with best practices and design patterns.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Brecher C, Rawat DB, Song H, Jeschke S (eds) (2017) Cyber-physical systems: foundations, principles and applications. Academic Press, London
2. Vogel-Heuser B, Fay A, Schaefer I, Tichy M (2015) Evolution of software in automated production systems: challenges and research directions. JSS 110:54–84
3. Werner B (2009) Object-oriented extensions for IEC 61131–3. IEEE Ind Electron Mag (IEEE Industrial Electronics Magazine) 3:36–39
4. Neumann E-M, Vogel-Heuser B, Fischer J, Ocker F, Diehm S, Schwarz M (2020) Formalization of Design patterns and their automatic identification in PLC software for architecture assessment. IFAC-PapersOnLine 53:7917–7924
5. Medvidovic N, Taylor RN (2010) Software architecture. In: Kramer J, Bishop J, Devanbu P, Uchitel S (eds) Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering—ICSE '10. ACM Press, New York, New York, USA, p 471
6. Reussner RH, Schmidt HW, Poernomo IH (2003) Reliability prediction for component-based software architectures. JSS 66:241–252
7. Bass L, Clements P, Kazman R (2010) Software architecture in practice, 2nd edn. Addison-Wesley, Boston (**14. print**)
8. Gamma E, Helm R, Johnson R, Vlissides J (2011) Design patterns: elements of reusable object-oriented software, 39. Printing. Addison-Wesley, Boston
9. Lytra I, Carrillo C, Capilla R, Zdun U (2020) Quality attributes use in architecture design decision methods: research and practice. Computing 102:551–572
10. Vogel-Heuser B, Fischer J, Feldmann S, Ulewicz S, Rösch S (2017) Modularity and architecture of PLC-based software for automated production Systems: an analysis in industrial companies. JSS 131:35–62
11. Prähofer H, Angerer F, Ramler R, Grillenberger F (2017) Static code analysis of IEC 61131–3 programs: comprehensive tool support and experiences from large-scale industrial application. IEEE TII 13:37–47
12. Nair A (2012) Product metrics for IEC 61131–3 languages. In: Proceedings of the 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012), vol 17, pp 1–8
13. Maga C, Jazdi N, Göhner P (2011) Reusable models in industrial automation: experiences in defining appropriate levels of granularity. IFAC Proc Vol 44:9145–9150
14. Clements P, Garlan D, Little R, Nord R, Stafford J (2003) Documenting software architectures: views and beyond. In: Proceedings 25th International Conference on Software Engineering, vol 25, pp 740–741
15. Güttel K, Weber P, Fay A (2008) Automatic generation of PLC code beyond the nominal sequence. In: Proceedings of the 13th International Conference on Emerging Technologies & Factory Automation (ETFA 2007). IEEE, vol 13, pp 1277–1284
16. Vogel-Heuser B, Fischer J, Hess D, Neumann E-M, Wurr M (2022) Boosting Extra-Functional Code Reusability in Cyber-Physical Production Systems: The Error Handling Case Study. TETC 10:60–73
17. Neumann E-M, Vogel-Heuser B, Fischer J, Keller J, Weis I, Diehm S, Schwarz M, Englert T, Stoll M, Zell U (2020) Identifying runtime issues in object-oriented IEC 61131-3-compliant control software using metrics. IEEE IECON 2020:259–266
18. Martin RC (2012) Clean code: A handbook of agile software craftsmanship, [Repr.]. Prentice Hall, Upper Saddle River
19. Vogel-Heuser B, Fischer J, Neumann E-M (2020) Goal-lever-indicator-principle to derive recommendations for improving IEC 61131–3 control software. In: 14th IEEE International Conference on Industrial Engineering and Engineering Management (IEEM 2020), vol 14, pp 1131–1136
20. VDMA (2020) Food Processing and Packaging Machinery. https://nuv.vdma.org/en/ueber-uns. Accessed 17 Apr 2022