Analysis and Evaluation of Variety Management in Software Defined Architectures

Kerstin Rossmann[1,*], Markus Berschik[1], Dieter Krause[1]

[1] Institute for Product Development and Mechanical Engineering Design, Hamburg University of Technology (TUHH)

** Korrespondierender Autor:*
Kerstin Rossmann
Denickestraße 17(Gebäude L)
D-21073 Hamburg
☎ *016093581827*
✉ *kerstin.rossmann@tuhh.de*

Abstract

The shift toward Software Defined Vehicles (SDV) is fundamentally changing vehicle development. Functions are increasingly implemented in software, updated over the air, and extended independently of hardware. Variant management is a key success factor to handle the resulting variety in development, production and operation. With the transition to SDV however, established variant management methods, traditionally oriented toward hardware-centric product architectures, are challenged by requirements arising from software-defined architectures. While recent research addresses the impact of the architectural shift in various areas of product development, there is limited research on the architectural characteristics resulting in new requirements for variant management in SDV in theory and practice. In this paper a systematic literature review on the characteristics of SDV is conducted to define evaluation criteria for the applicability of variant management methods in software-defined architectures.

Keywords

Software Defined Vehicle, Variant Management, Software Defined Architecture

 | DOI: 10.35199/dfx2026.27

1. Motivation

The automotive industry is currently undergoing as fundamental architectural shift driven by the Software Defined Vehicle (SDV) paradigm. A central characteristic of this shift is the decoupling of software from hardware. By separating software development and deployment from hardware lifecycles manufacturers aim to enable higher rates of over-the-air (OTA) updates, support a continuously growing number of retrofittable features, and thereby improve internal hardware development efforts [1].

At the same time, software functions are still often strongly tied to individual electronic control units (ECU). This close hardware-software coupling contradicts the objective of software-defined architectures, in which software increasingly becomes responsible for security, functional improvements, and feature enhancements throughout the product lifecycle. The flexibility required for this transformation is further constrained by the increasing complexity of interfaces and dependencies between software functions, hardware and system configuration [2]. Hence, a single vehicle generation may run on a combinatorial matrix of hardware generations, software versions, and activated features, with each combination constituting a distinct configuration variant. The resulting variety is therefore no longer a static product configuration problem, but a dynamic product lifecycle challenge. This is a structural difference compared to traditional vehicle architectures, reflected in Figure 1 and reveals limitations in the applicability of existing requirements for variant management [3].

In this paper a systematic literature analysis is conducted to assess relevant characteristics of SDV with implications on variant management. Based on these characteristics explicit requirements are derived to exemplarily assess the applicability of the defined requirements on a set of variant management methods.

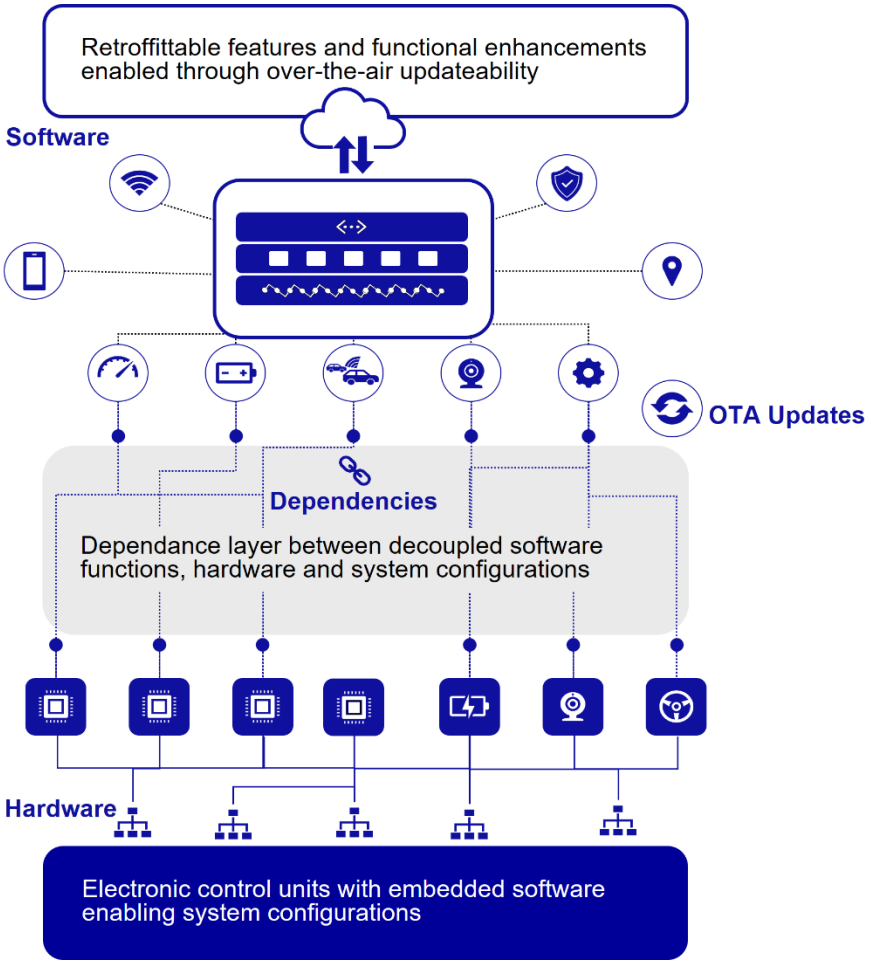

Figure 1: Simplified Layers of Software Defined Architectures

2. Background

In this section the theoretical background is introduced. Specifically, the impact of trends in the automotive industry in the context of product architecture and variant management is addressed.

2.1. Architecture Definition

In product development, a product architecture is used for the structural description of products. This encompasses the product structure, that is, the physical structure and the functional structure, which includes the functional description of a product [4].

The product structure describes the physical and hierarchical organization of a product, similar to the structure of a bill of materials. Product structures use assemblies and parts to describe the structure. Depending on the context, however, the use of components may also be relevant [5]. Components are not traditionally included in bills of materials. However, they can be used to break down the structure more effectively by applying different levels of abstraction. Functional structures represent hierarchical or flow-based relationships among functions. The hierarchical representation illustrates the division of functions into subfunctions. The flow-oriented functional structure uses flows between functions to describe a product's behavior. Taken as a whole, the architecture represents the structural and functional description of a product. The exact design of the architecture depends on the development domain [6].

The automotive industry traditionally applies hardware-defined architectures in which the software and functionality is bound to one ECU, often referenced as one component. The increasing demand for software functions therefore results in increasing numbers of ECUs in distributed architectures. In this architecture further configuration is achieved through parametrization and dataset adaptions.

2.2. Variant Management

The goal of variant management is to improve the variety of offerings while keeping the impact on internal costs manageable. Expanding the product portfolio often requires more components and processes. This inevitably increases complexity within the company. Reducing the complexity caused by variants is therefore another key task of variant management. Variant management thus encompasses the design, steering, and control of variant variety as a cross-functional issue within the company [6].

Fundamentally, variant management comprises four areas of responsibility. Variant generation involves designing new product variants without allowing variance to become excessive. At the same time, variants may not be generated in the first place. This is described under variant avoidance. If variants are not relevant or demanded by the market, they should not be developed from a variant management perspective, or, if already in existence, they should be discontinued. The goal should be variant control, in which the required external variety can be achieved with as little internal variety as possible. The processes are optimized for variance as much as possible. If this state has not yet been achieved, variant reduction should be employed. This process eliminates unnecessary variance from the structure and processes [7].

In practice the focus is often on the ability to document internal variety and ensure consistent configuration management across variants, resembling the external variety available to the customer as specific product configuration. In hardware defined architectures a common approach is to determine the technically feasible product configuration with a set of components and variant coding constituting a specific product variant. The underlying rule database is based on Boolean algebra and allows to manage variants by attaching

configuration rules to each component. The variants are documented in hierarchical structures, ofttimes resembled as variant bill of material [8].

3. Research Objective

Software-defined architectures challenge existing variant management methods, as they introduce architectural characteristics that extend the criteria applicable for traditionally hardware-oriented product development [9]. In particular, the decoupled evolution of hardware and software over the product lifecycle results in a dynamic set of configuration variants that cannot be sufficiently addressed as static product configurations [3]. Instead, software-oriented products require methods that can represent and manage variants across different architecture layers, artifacts, and lifecycle phases.

In science and practice, several approaches for modeling, documenting, and managing variants already exist. Established methods for variant management already support common requirements for traceability, restrictions, and interdisciplinarity and enable the derivation, reuse, and adaption variants. However, these approaches have mainly been designed for product development environments with static sets of configuration variants and predefined restrictions [10].

The extent to which these methods meet requirements of software-defined architectures has not yet been sufficiently investigated.

This leads to the following research questions:

- RQ1: Which requirements need to be considered to evaluate the applicability of variant management methods in software-defined architectures?
- RQ2: To what extend do existing variant management methods meet specific requirements in the context of software-defined architectures?

The objective of this research is to derive requirements from the characteristics of software-defined architectures and to analyze and evaluate existing variant management methods regarding their applicability these highly dynamic, variant-rich architectures. The assessment primarily aims to validate the identified requirements and reveal potential gaps where existing approaches do not provide sufficient support, thereby motivating the need for further research.

4. Research Approach

In this paper a systematic literature review is conducted to identify contributions describing characteristics of software-defined architectures in the automotive context. These characteristics reflect the differentiating attributes of hardware-centric and software-centric architectures. This step provides the basis for understanding the impact of the architectural transformation using the example of SDV.

Secondly, requirements for variant management in software-defined architectures are derived from these characteristics. These requirements are defined to support the ability of managing variety constituting the characteristics identified in literature. Establishing a specific set of requirements as evaluation basis for the applicability of variant management methods.

Finally, these requirements are exemplariliy applied to a set of variant management methods addressing the management of products with high variety or evaluated in automotive. The purpose of this assessment is to verify the applicability and relevance of the derived requirements. Furthermore, the evaluation results are consolidated to identify strengths and limitations of existing methods and derive existing research gaps.

5. Results

The systematic literature review identified relevant articles addressing the implication of the architectural shift toward software-defined architectures. To expand the coverage of the literature analysis, a manual exploratory search was carried out. Thereby further relevant contributions were identified. The search results are the basis for the identification of the relevant characteristics and the derivation of requirements for variant management in software-defined architectures.

5.1. Search String

Based on the research questions the search string was formed consisting of key words in three concept groups, connected by boolean operators. Within each group the search terms are connected using “OR”, and wildcard characters (*) are applied to capture term variations (Table 1).The first group addresses the architectural shift towards software defined architectures and sets the automotive context, the second group further specifies the area of interest, in this case variant management and software, and the third focuses on challenges and activities related to variant handling, such as traceability, compatibility and lifecycle management. This search string was applied in the Scopus-database to cover a variety of databases like IEExlore and ACM.

Table 1: Search String

("software defined vehicle" OR "software defined architecture" OR "automotive") AND ("variability" OR "varia* management" OR "software documentation" OR "requirements") AND ("software varia*" OR " varia* handling" OR "dependence management" OR "update traceability" OR "compatibility management" OR "architecture documentation" OR "software only features" OR "over the air" OR "product lifecycle" OR "product life phase*")

5.2. Selection Criteria

Following the PRISMA method criteria for the screening on title and abstract as well as fulltext were defined [11]. Before the screening additional filters were set to refine the search regarding published journal and conference proceedings in English or German, associated with the subject are of engineering or computer science, resulting in 840 relevant publications.

During the title and abstract screening phase articles were excluded if they did not align with the scope of this review. Specifically, records were removed when they were not related to automotive, did not address SDV, did not address architectural aspects, or primarily focused on hardware-defined architectures. The remaining 75 articles underwent a full-text screening to assess their relevance in greater detail. Records were excluded if they focused on security implementation criteria, secure update methods, specific technical solutions, or discussed IT architecture implications, limiting the number to 17 publications. From the manual exploratory search 4 relevant records were added resulting in 21 relevant records included in this analysis. The stages of the search process and exclusion criteria are described in Figure 2.

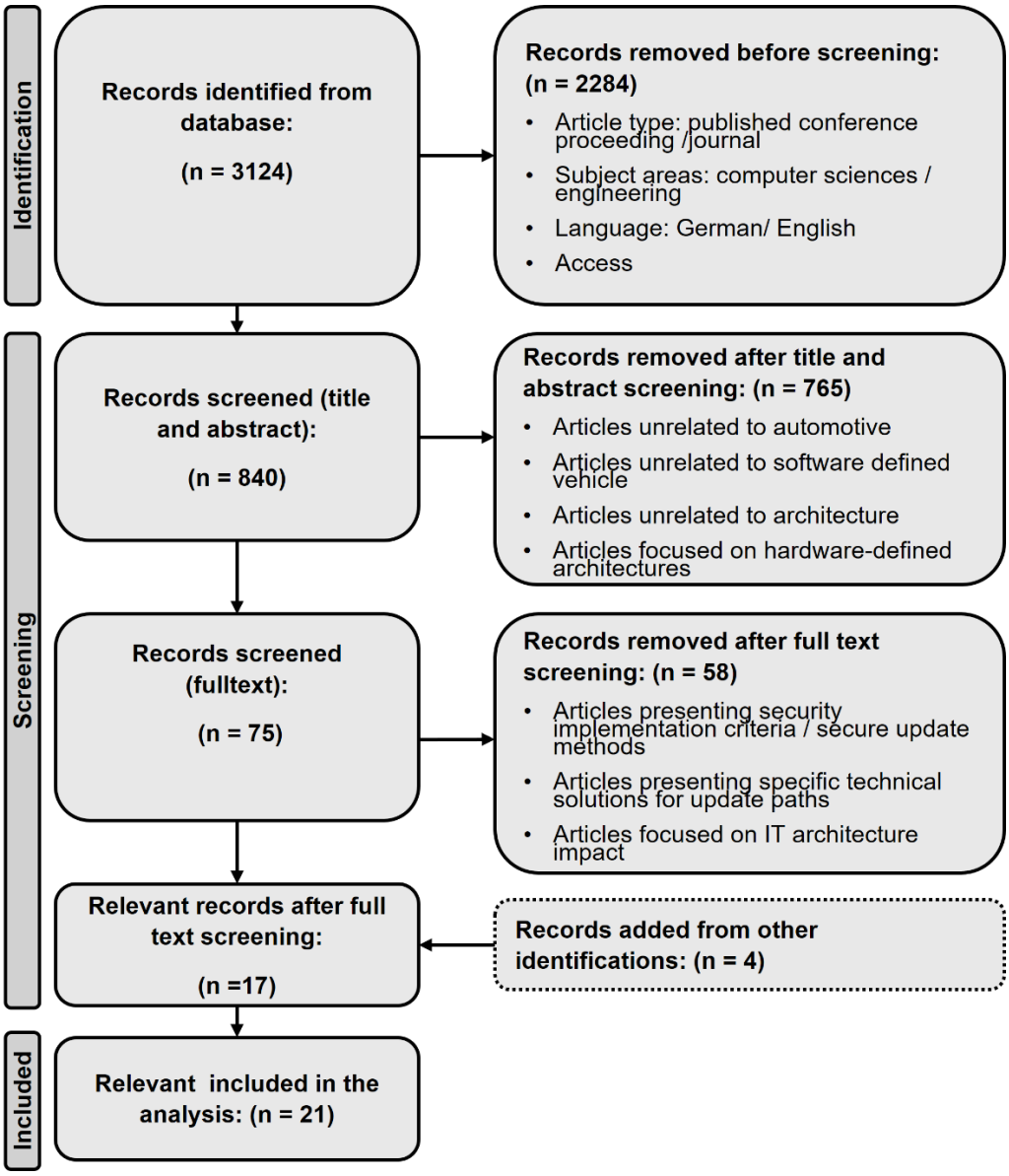

Figure 2: Stages of the Search Process with Exclusion Criteria

6. Results

The research shows software-defined architectures inherit characteristics, especially hardware-software decoupling and frequent software updates, that need to be explicitly considered in variant management. Leading to the assumption that software-defined architectures require variant management methods which address additional requirements like dependencies across architectural layers or lifecycle-oriented impact assessment which can be derived from the characteristics of SDVs.

6.1. Requirements for Variant Management in Software Defined Architectures

Based on the literature review thre relevant characteristics of software-defined architectures which need to be considered for deriving requirements for variant management can be identified from the automotive usecase of SDV. These characteristics include the decoupled evolution of software and hardware, lifecycle updateability through over-the-air (OTA) updates and the integration of cloud platforms and services [2], [12], [13]. Together, these characteristics fundamentally change how variants are created, deployed, maintained, and

evolve throughout the lifecycle. Thereby each characteristic entails new requirements for variant management.

The decoupled evolution of software and hardware enables software functions to evolve independently from hardware generations and vehicle platforms [13]. Meaning, software-defined functionality increasingly emerges from software components deployed on application neutral hardware platforms, resulting in variants at software and hardware level [12]. This requires support for the distinction of **variant types**, enabling the differentiation and separate management of software, hardware and coupled variants. Furthermore, as software and hardware evolve independently while remaining functionally dependant, variant management must support **vertical traceability**, allowing dependancies and interfaces to be traced across architectural layers.

Another characteristic of software-defined architectures is the continuous evolution of software during operations, often described as lifecycle updateability. This allows the continuous modification of product functionality through OTA updates supporting the deployment of new features, functional software releases and bug fixes [14], [15]. As software-defined functionality evolves throughout the operational lifetime of a product, variant management must support **lifecycle traceability** to assess variant impact across development, production, operation, and maintenance. The increasing frequency of software releases further requires **scalability** and must enable the efficient management of growing numbers of releases, updating or extending the functionality. In addition, variants may be triggered by different causes, i.e. optional functional enhancements, or corrective modifications driven by environmental changes or regulatory adaptions [16]. Consequently, variant management shall support the identification of **variant drivers**, to allow the identification of causes for variant creation. Lifecycle updateability requires the consideration of an increasing number of technical constraints. Variants must comply with architectural restrictions, safety considerations, and security constraints in order to ensure the intended functionality of a product [3]. This is reflected in the requirement **internal restrictions**, which shall address the representation and management of technical constraints. In addition to internal restrictions the implications of highly regulated environments need to be considered in order to ensure the technical compliance of a product [17]. The requirement **external/contextual restrictions** shall reflect the resulting implications of external constraints on variant creation and applicability.

The increasing integration of cloud platforms and services further extends the adaptability of functions beyond the physical product and enables dynamic interactions between onboard software, backend systems, and digital services. As a result the adaption of product behavior based on user preferences, available services, or operational contexts is possible [2], [12]. Therefore, variant management must support **continuous reconfiguration**, enabling the management of temporary and context- dependant configurations that can dynamically change during operation.

Together the requirements summarized in Table 2 address the characteristics of SDVs relevant for variant management. They provide a structured basis for analyzing existing variant management methods and assessing their applicability in the development and operation of software-defined architectures.

Table 2: Description of Requirements for Variant Management in Software-Defined Architectures

Description of Requirements for Variant Management
Variant Types [2], [9], [12]: Distinguish and manage hardware, software and coupled variants as separate variant categories
Vertical Traceability [2], [3]: Enable the assessment of dependencies and interfaces between software variants, hardware variants and associated system functions

Lifecycle Traceability [3], [14], [15]: Ensure the assessment and traceability of variant impact across development, deployment, operation, maintenance and update activities throughout the product lifecycle

Internal Restrictions [3], [17]: Capture, maintain and validate restrictions imposed by architecture, functional safety concepts, security mechanisms and compatibility constraints

External/Contextual Restrictions [16], [17]: Consider regulations, legal constraints, market-specific requirements, and operational conditions that limit or influence the applicability of variants

Variant Driver [15]: Ensure the identification and differentiation of drivers for variant creation, enabling the distinction between variants resulting from environmental changes, regulatory adaptions, corrective actions, and functional enhancements

Scalability [2]: Effectively manage continuously growing variant spaces resulting from frequent software releases

Continuous Reconfiguration [13], [14]: Support the management of temporary and context-dependant variant configurations based on user preferences, available services or environmental conditions

6.2. Evaluation of Variant Management Methods

To verify the applicability of the derived requirements for evaluating variant management in the context of software-defined architectures, the requirements are applied to a set of methods (Figure 3). The evaluation serves as exemplary application of the derived requirements and illustrates how these can be used to systematically analyze existing variant management methods.

The analysis indicates the requirements **internal restrictions** and **scalability** are comparatively well addressed across the evaluated methods. Commonly mechanisms are provided to represent dependencies between variants and define valid product configurations through variability constraints or feature interactions across domains [18], [19]. Scalability is i.e. addressed through architectural or functional decomposition or integrated feature models which support the management of large variant spaces.

Requirements such as **variant types** and **vertical traceability** reveal clear differences among the methods. While interdisciplinary methods explicitly consider software, hardware, and their dependencies, model-based approaches primarily focus on a generic variability representation without the distinction in variant types [19]. The support for dependencies between software, hardware and system artifacts is mainly addressed by interdisciplinary methods and architecture-oriented approaches [18], [20], [21]. Lifecycle traceability is in general supported by the same characteristic, allowing the management of reuse and product configuration. However, the support for assessing and tracing variant impact beyond development, to operation and maintenance is less explicitly represented.

The requirements **external and contextual restrictions** and **variant drivers** provide additional differentiation between the analyzed methods. While customer requirements are commonly represented as source of variants, only limited support can be identified for systematically distinguishing between variant drivers such as regulatory, corrective or functional drivers [18], [20]. Similarly, the explicit consideration of external or contextual influences such as market specifics or regulatory requirements are primarily found in the method proposed by von Esebeck et al.

The most significant gap concerns the requirement of continuous reconfiguration. The analyzed methods predominantly focus on variant management in the development under consideration of software configuration [18], [19], [20]. This is extended by the integration of software evolution through over-the-air updates [21]. However, temporary, context-dependant dynamic reconfiguration cannot be managed consistently across the analyzed methods.

The exemplary application of the requirements for variant management in software-defined architectures demonstrates the proposed requirements can be applied consistently across different variant management methods. The analysis indicates variant management methods support requirements also prevalent in hardware-defined architectures, like compatibility and dependency management, reflected in traceability or scalability. However, other characteristics attributed to software-defined architectures like lifecycle traceability, contextual restrictions and continuous reconfiguration remain only partially addressed or not supported.

Figure 3: Requirements for Software-defined Architectures applied on Variant Management Methods

7. Summary and Discussion

In this paper requirements for variant management in the context of software-defined architectures are proposed. As a basis for the derivation of these requirements a systematic literature review on the characteristics of SDV with relevancy for variant management was conducted. Through the exemplary application of the derived requirements on a set of variant management methods their vailidity and applicability was initially verified. The comparison of the variant management methods demonstrates that the derived requirements address several aspects which are only partially covered by existing approaches. In particular, the differences indicating the increasing separation of software and hardware and the growing importance of software evolution along the lifecycle introduce requirements which are currently not consistently addressed. Simlarly the varying consideration of external restrictions, variant drivers, and continuous reconfiguration suggests that these requirements represent important dimensions for establishing variant management in software-defined architectures.

Overall, the exemplary application of the requirements demonstrates their applicability for the evaluation of variant management approaches in the context of software-defined architectures, indicating the requirements are not consistently covered by existing methods. i.e. lifecycle traceability, the structured differentiation of software and hardware variants, and software evolution emerge as important evaluation dimensions. These observations support the relevance of the derived requirements and provide a foundation for development of a variant management framework adapted to the characteristics of software-defined architectures.

Literaturverzeichnis

[1] Z. Liu, W. Zhang, and F. Zhao, ‘Impact, Challenges and Prospect of Software-Defined Vehicles’, *Automot. Innov.*, vol. 5, no. 2, pp. 180–194, Apr. 2022, doi: 10.1007/s42154-022-00179-z.

[2] A. Khamis and P. Goswami, ‘Rethinking Vehicle Architecture Through Softwarization and Servitization’, *IEEE Access*, vol. 13, pp. 126213–126226, 2025, doi: 10.1109/ACCESS.2025.3588432.

[3] X. Zhu, R. Sturm, C. Seiler, and S. Wagner, ‘Complexity Handling in the Software-Defined Vehicles: Documenting the Expert Knowledge’, in *2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C)*, Odense, Denmark: IEEE, Mar. 2025, pp. 553–556. doi: 10.1109/ICSA-C65153.2025.00082.

[4] K. T. Ulrich and S. D. Eppinger, *Product Design and Development*, Sixth edition. New York, NY: McGraw-Hill Education, 2016.

[5] G. Pahl, W. Beitz, J. Feldhusen, and K.-H. Grote, *Pahl/Beitz Konstruktionslehre: Grundlagen erfolgreicher Produktentwicklung. Methoden und Anwendung*, 7. Aufl. 2007. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[6] D. Krause, *Methodical Development of Modular Product Families: Developing High Product Diversity in a Manageable Way*, 1st ed. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2023.

[7] H. Wildemann, *Variantenmanagement: Leitfaden zur Komplexitätsreduzierung, -beherrschung und -vermeidung in Produkt und Prozess*, 26. Auflage. in Leitfaden / TCW Transfer-Centrum für Produktionslogistik und Technologie-Management, no. 5. München: TCW-Verlag, 2018.

[8] M. Käßmeyer, M. Schulze, and M. Schurius, ‘A process to support a systematic change impact analysis of variability and safety in automotive functions’, in *Proceedings of the 19th International Conference on Software Product Line*, Nashville Tennessee: ACM, Jul. 2015, pp. 235–244. doi: 10.1145/2791060.2791079.

[9] F. Pan *et al.*, ‘Towards Software-Defined Vehicles: From Model-based Engineering to Virtualization-based Deployment’, *IEEE Access*, pp. 1–1, 2024, doi: 10.1109/ACCESS.2024.3512002.

[10] B. Menninger *et al.*, ‘Modeling and analysis of functional variance of complex technical systems’, in *DS 119: Proceedings of the 33rd Symposium Design for X (DFX2022)*, The Design Society, 2022, pp. 10–10. doi: 10.35199/dfx2022.15.

[11] D. Moher, A. Liberati, J. Tetzlaff, D. G. Altman, and for the PRISMA Group, ‘Preferred reporting items for systematic reviews and meta-analyses: the PRISMA statement’, *BMJ*, vol. 339, no. jul21 1, pp. b2535–b2535, 2009, doi: 10.1136/bmj.b2535.

[12] A. Bazzi, A. Shaout, and D. Ma, ‘A Novel Variability-Rich Scheme for Software Updates of Automotive Systems’, *IEEE Access*, vol. 12, pp. 79530–79548, 2024, doi: 10.1109/ACCESS.2024.3409629.

[13] H. Askaripoor, M. Hashemi Farzaneh, and A. Knoll, ‘E/E Architecture Synthesis: Challenges and Technologies’, *Electronics*, vol. 11, no. 4, p. 518, 2022, doi: 10.3390/electronics11040518.

[14] L. Holsten, J. Krüger, and T. Leich, ‘Insights into Transitioning towards Electrics/Electronics Platform Management in the Automotive Industry’, in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, Porto de Galinhas Brazil: ACM, 2024, pp. 161–172. doi: 10.1145/3663529.3663837.

[15] H. Guissouma, C. P. Hohl, F. Lesniak, M. Schindewolf, J. Becker, and E. Sax, ‘Lifecycle Management of Automotive Safety-Critical Over the Air Updates: A Systems Approach’, *IEEE Access*, vol. 10, pp. 57696–57717, 2022, doi: 10.1109/ACCESS.2022.3176879.

[16] D. F. Blanco, F. Le Mouël, T. Lin, and M.-P. Escudié, ‘A Comprehensive Survey on Software as a Service (SaaS) Transformation for the Automotive Systems’, *IEEE Access*, vol. 11, pp. 73688–73753, 2023, doi: 10.1109/ACCESS.2023.3294256.

[17] A. Bucaioni, P. Pelliccione, and S. Mubeen, ‘Modelling centralised automotive E/E software architectures’, *Adv. Eng. Inform.*, vol. 59, p. 102289, Jan. 2024, doi: 10.1016/j.aei.2023.102289.

[18] B. Vogel-Heuser, J. Fuchs, S. Feldmann, and C. Legat, ‘Interdisziplinärer Produktlinienansatz zur Steigerung der Wiederverwendung: Untersuchung der Anwendbarkeit von Produktlinien für die Entwicklung in der Automatisierung des Maschinenbaus’, *- Autom.*, vol. 63, no. 2, pp. 99–110, 2015, doi: 10.1515/auto-2014-1140.

[19] T. Weilkiens, *Variant modeling with SysML*. in MBSE4U booklet series. Fredesdorf: MBSE4U, 2016.

[20] T. Schulte, T. Dickopf, and A. Standke, ‘Integration & CTP-Spezialisierung’, in *Modellbasierter Entwicklungsprozess cybertronischer Systeme*, M. Eigner, W. Koch, and C. Muggeo, Eds, Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 87–92. doi: 10.1007/978-3-662-55124-0_10.

[21] R. von Esebeck, Y. Lindebauer, and T. Vietor, ‘Variantenkomplexität in der automobilen Steuergerätekonfiguration: Status quo und Potenziale durch SPLE’, in *DS 140: Proceedings of the 36th Symposium Design for X (DFX2025)*, The Design Society, 2025, pp. 199–208. doi: 10.35199/dfx2025.21.