22nd INTERNATIONAL DEPENDENCY AND STRUCTURE MODELING CONFERENCE, DSM 2020
CAMBRIDGE, MASSACHUSETTS, USA, 13 – 15 October, 2020

A Matrix-Based Blueprint for System Architecture Design – A Case Study with an Industrial Partner

Benjamin Lender[1], Jakob Trauer[2], Sebastian Schweigert-Recksiek[2], Karsten Spreitzer[3], Nils Chmielewski[3], Markus Zimmermann[2]

[1]Laboratory for Machine Tools and Production Engineering WZL of the RWTH Aachen University
[2]Laboratory for Product Development and Lightweight Design, Technical University of Munich
[3]Viessmann Werke Allendorf GmbH

Abstract: Trends like digitalization, servitization and the ongoing globalization drive an ever more growing product complexity. To handle this challenge and to succeed on fast-paced markets, companies need to constantly model, organize, and improve their system architectures. This paper combines different approaches from the literature with specific needs of the industry partner. An overview of definitions and processes found in literature is the basis for deriving a generic framework that enables companies to implement a system architecture design process that builds the basis for further actions like risk management or the introduction of platform strategies. The main part of the framework is a guideline on how to ensure a comprehensive system architecture regardless of the amount of resources spent on the process. The framework is applied to different use cases of the industry partner at varying levels of complexity.

Keywords: system architecture design, functional architecture, physical architecture, functional modeling, product development process

1 Introduction

The necessity to deal with complexity increases with the rise of individualized products at mass scale, as pointed out by Fuchs and Golenhofen (2019). The choice of a system architecture is a strategic decision as pointed out by Ulrich (1995) and Maurer (2007), who consider it a key to addressing the aforementioned challenges. In this context, Crawley et al. (2016) underline that complexity is in itself neither good nor bad. Increasing efficiency of a product in response to ever-increasing demands comes with increased complexity.

System architectures as the conceptual structure of a product or system can grow in iterations and become increasingly integral instead of modular (Fuchs and Golenhofen, 2019). Maurer (2007) states, that even though products may implicitly have an architecture, it is common for the architecture not to be documented or explicitly known. This however is the basis for profiting from the potential architectures have for managing complexity (Weilkiens et al., 2015). Wise et al. (2015) show an example of possible consequences of non-explicitly documented architectures. They are tasked with re-architecting a legacy avionics system. Half of their process covers understanding the system's undocumented architecture.

This paper presents a framework for system architecture design processes that offer means to validate a resulting system architecture. It is aimed at making it easier for companies to design and validate their system architectures.

2 State of the Art and Research

In the literature, there is no unified understanding of what an architecture in product development is. The INCOSE handbook describes the architecture as "the basic concept or properties of a system in its environment, realized through its elements, relationships and principles of design and development" (Walden et al., 2017). Marti (2007) argues, that the system architecture "consists of the structure of functionality, the structure of physical components, and the mapping from functionality to physical components". This definition of system architecture is also found with Luzeaux and Wippler (2017) and Crawley et al. (2016). The latter definition has the advantage of being about what objects make up the system architecture, leaving out principles, which are hard to grasp. Following this definition, it is possible to determine, whether a system architecture is complete. A complete system architecture is made up of a functional architecture, being the structure of function, the physical architecture, being the structure of the physical elements also called "form", and a mapping from function to form. As Crawley et al. (2016) point out; the transition from function to form is the transition from solution neutral to solution specific implementation. A solution and technology neutral functional architecture can be stable over product generations. The physical architecture can apply evolving technologies to carry out a certain function. This underlines the importance of the functional architecture: it brings stability without inhibiting innovation (Fuchs and Golenhofen, 2019).

According to Crawley et al. (2016), the physical architecture "includes the entities of form and the formal relationships among the entities." The functional architecture is defined by Weilkiens et al. (2015) as an "architecture based on functional elements, functional interfaces and architecture decision". This matches the definition of the physical architecture, except for architecture decisions. At this point a definition of function needs to be chosen, as a literature review by Erden et al. (2008) identifies as many as 17 possible definitions. This contribution will define a function according to Lindemann (2009) as "a purpose-oriented, solution-neutral relation between input and output, stated in form of an operation". This means that "raising a temperature" is a function, while "raising a temperature in four seconds by 10 °C" is not. Architecture decisions account for such non-functional requirements that are relevant for a functional architecture.

To design architectures, the processes found in the literature focus on conveying an understanding of the architecture. The function-concept-form (FCF) paradigm as proposed by Crawley et al. (2016) is at the heart of the processes considered here, as it describes the basic approach behind the system architecture definition followed by this paper. An approach coming from design is function-behavior-structure (FBS) by Gero (1990). The FBS framework starts with the function from which an expected behavior is deducted. A structure is then designed to execute the expected behavior. Comparing the behavior of the structure to the expected behavior forms the basis for future iterations, resulting in the design. The expected behavior is planned, while the behavior of the structure is observed.

Luzeaux and Wippler (2017) propose the G3CF2B as seen in Figure 1, which combines the FCF paradigm with the FBS. G3CF2B stand for goal, capabilities, contexts, concept, form, expected behavior and emergent behavior, where the numbers reference how often the initial letter appears. It adds the behavioral aspects of FBS to the FCF. Increasing the applicability, it also models the goal, the context, and how they in turn influence the architecture.

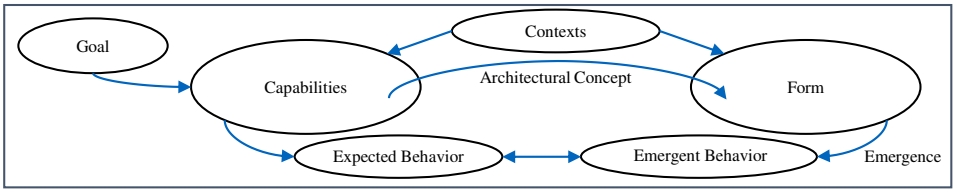

Figure 16: G3CF2B according to Luzeaux and Wippler (2017)

3 Research Methods and Approaches

The project was carried out according to the design research methodology proposed by Blessing and Chakrabarti (2009) as a type five project, going through the phases of research clarification, descriptive study I, prescriptive study and descriptive study II. The project focused on an assessment of the current situation at the industry partner, comparing it to the state of art and research. The research question was “How can a systematic approach for system architecture design be integrated into the product development process of an international climate solution company?” Matching an overview of the literature to the experiences of the industry partner resulted in the presented blueprint framework. Instead of identifying possible gaps of the process at the industry partner as a starting point, it was decided to focus on building a new blueprint, as no approach was found in literature to create processes for companies with different levels of resources that can ensure the resulting system architecture to be complete.

Working with the industry partner provided the opportunity to implement parts of the blueprint as a case study, rather than working purely literature-based. The Viessmann Group is a family owned international climate solution company of more than 12,000 employees around the world, producing climate solutions for applications ranging from domestic to industrial applications. The case study was carried out with a heating system for a single apartment, modeling parts of the software implementation, and the combustion and heating process. The case study focused exclusively on the step from the expected behavior to the functional architecture, as this was an area of interest for the industry partner.

4 Blueprint

4.1 Common Understanding of the Blueprint

In contrast to the generic processes found in literature, the blueprint is not a process designed to be implemented directly in the given form. As the name suggests, it is rather a template that must be filled out to derive a system architecture design process. It is flexible regarding the methods to be used, but offers guidelines on how to match the components. This aims at making it easier for companies to integrate the blueprint into existing development environments. It also serves to examine whether every aspect of the blueprint is covered in one way or another by the existing architectural processes of a company, potentially identifying gaps that may lead to problems handling complexity and avoid long rework cycles. The created artifacts also serve as documentation of the process.

4.2 Overview

An overview of the blueprint is shown in Figure 2. The grey area marks the core of the system architecture, consisting of functional architecture, concept and physical architecture. In addition, there is the behavioral aspect at the bottom, the initial goal on the left and the context of the system at the top. On the right-hand side, there are various optional follow-up actions. Large arrows indicate a suggestion for steps of the resulting process. Small arrows indicate control mechanisms.

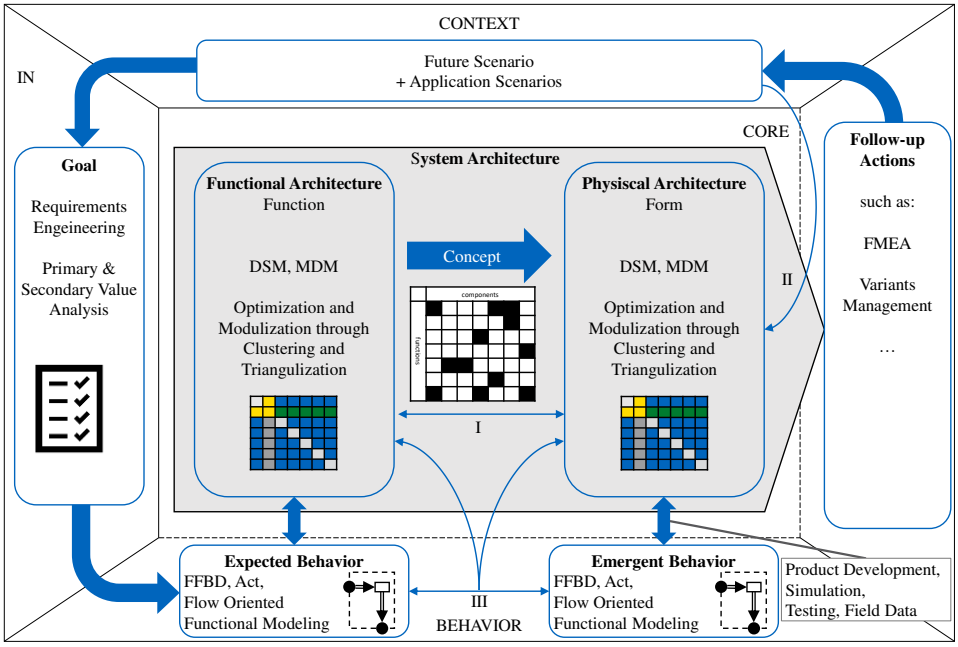

Figure 17: Overview of the blueprint

The basis for any application of the blueprint are context and goal. The context contains any information that is not part of the system but affects the system. This includes information on local laws, the company's business model regarding the specific product/project, or information on environmental conditions in the area of deployment. This information will in turn be the basis for the requirements, which are part of the "goal". The goal contains any information about what the product should do or be. Therefore, it encompasses the requirements engineering. Stating the goal from a customer's perspective leaves room for the technical development side to not constrain the solution space too early. However, this must not omit other requirement sources such as regulations or special climate conditions.

Based on the goal, an expected behavior is formulated at the interface of product management, representing the customer and/or the company, and the engineers responsible for the technical implementation. Explicitly modeling the expected behavior proved to be a key for effective discussion about the system. Both engineering and product management are discussing the same dynamic behavior of the system. Note that while architectures are strictly static, behavior is strictly dynamic (Lamm and Weilkiens, 2010). As in FBS, the emergent behavior is exhibited by the actual form of the system, be it as a simulation, a prototype in testing, or a product deployed in the field. Both behaviors must be modeled the same way or in compatible ways in order to compare them. Comparing a SysML act to an excel sheet of field data will not yield the desired insights.

Building on the expected behavior, the functional architecture of the system is documented, an example of which is shown in the case study. The Functional Architecture for Systems method (FAS) by Weilkiens et al. (2015) has been chosen for the case study, but any method is applicable. FAS lists all steps of the behavior, translates them to functions and groups these functions. Based on the functional groups and the relations between the functions, the architecture is built. In an early phase, an architecture is likely to contain only desired functions. With increasing experience, awareness of undesired functions will increase, which must be included in the model. This is necessary to model change propagation. In this way, the functional architecture can support the knowledge management of a company. It is common for knowledge to amount over years of working experience and to be spread over multiple engineers. That way, the knowledge is susceptible to getting lost when an employee leaves the company.

From the functional architecture, one or several concepts lead to one or several physical architectures. Applicable methods are proposed in VDI 2221 (1993). As with the behavior, both types of architecture must be modeled the same way in order to be comparable. A physical architecture executing the functional architecture can change over product generations, while the functional architecture remains stable (Crawley et al., 2016). The arrow from the physical architecture to the emergent behavior of the structure contains the product development process. This is an issue of both the FBS and the G3CF2B: there is no behavior of a structure without a structure, as pointed out by Galle (2009). It is to be noted that documenting all requirements itself can take an extensive effort. The same applies to documenting the architecture itself. FBS and G3CF2B do not discuss this gap in architecture design. An approach to avoid not getting any architectural results for years while developing new products could be to focus on the top-level architecture and to

increase the level of detail from there. That way, it is possible to deliver working results for the architecture before finishing the last part of the documentation. As incorporated by Schuh et al. (2007), the suggested framework also entails follow-up actions, seen on the right-hand side of Figure 2. Follow-up actions use the entire system architecture for purposes that are not necessarily part of the product development yielding emergent behavior. These actions can include variant management, change management, risk management approaches like FMEA, or any action, that makes use of the system architecture besides product development itself. The list can be extended as necessary for a given application.

4.3 Control Mechanisms – Validating the Architectures

The thin arrows in Figure 2 mark the three control mechanisms of the blueprint. They are an essential feature and enable companies to validate existing or new system architectures at various points in the process. As has been stated previously, being able to compare the expected behavior to the behavior of the structure may take years of development and is not advisable as the only means of validating the architecture. Both control mechanism I and control mechanism II can be applied before that. Control mechanism I is a direct comparison of the functional and the physical architecture. Therefore, they must be modeled the same way. Expected differences in the matrices arise from the more stable functional architecture having a higher level of abstraction. Components might be related through physical structures that have no functional relation. Differences can be found by direct comparison of the two matrices, either through visual inspection or through a program, and rated by engineers. In case of critical differences, the concept must be adapted. This cycle can happen before any technical development has started.

Control mechanism II is comparing the physical architecture(s) to the context. This mechanism is useful to avoid concepts that are unsuitable for the business plan. An example from the climate solutions industry would be a heating system burning oil, aimed at a market in which such heating systems will not be allowed for sale within five years of start of production. In this case, the context can be documented as a fact sheet covering the market, which includes the deadline for a certain type of heating system. If the project yields a positive net outcome before that, it may be worth pursuing the architecture. Otherwise, it is better to abandon the architecture before more effort is spent on it. Other applications are the test for environmental conditions or the suitability for markets, considering that norms may differ strongly between markets.

Control mechanism III is feasible as soon as an emergent behavior has been observed and modeled in a way that can be compared to the expected behavior. This can take a considerable amount of time, hence the importance of control mechanisms I and II. In analogy to control mechanism I, control mechanism III directly compares the two sets of behaviors. Such differences might be e.g. unanticipated vibrations, causing inacceptable noise. Effects originating from physical links can be hard to anticipate without prototypes and will therefore need to be added to the functional architecture as knowledge increases with experience. Differences are evaluated based on their criticality for the product's functionality. A new iteration of architecture development is started according the origin of any critical difference. It is therefore advisable to generate real behavior early in the development process.

4.4 Handling Obstacles during the Implementation

Development does not stop for an existing architecture to be documented and optimized. Experience also shows that resources for this effort can be limited, due to economic constraints. The blueprint explicitly does not require certain tools to be used, only that in accordance with the control mechanisms certain tools and methods must match. An example is the use of the same method of modeling for both the functional and the physical architecture. The first step in any application of the blueprint is the modeling/documentation of the context and the goal. Without either one, one is unable to verify an architecture, as there is no explicit right or wrong. For products based on primary values (Crawley et al., 2016), which are the values the product exists for, the documentation can be accounted for by knowledge among experienced engineers. With increasing relevance of secondary values, this becomes infeasible. Tools must be chosen for each step of the blueprint, which can in the most lightweight example be “talk about it and draw a model on paper”. Most companies may already have tools in place for various steps that need to be considered when deciding on other tools. Figure 2 offers examples deemed suitable by the authors but cannot provide an exhaustive list of tools and methods.

5 Case Study

The case study focuses on the step from the expected behavior to the functional architecture. The limitation of scope is due to the process currently being implemented at the industry partner. A number of case studies have been carried out covering three software functions, two physical processes, and the combination of a software function and a physical process shown here. The FAS method has been chosen for implementation. While Weilkiens et al. (2015) suggest SysML, MDMs were chosen for the case study to handle the large amount of information. Both methods are possible; the blueprint does not restrict itself to one or the other. Figure 3 shows the behavior in a flow oriented functional model (Lindemann, 2016). The model shows both the user interaction on the left and the combustion process on the right side, covering each intermediary condition. For modelling programs, this decreases the overview. The conditions assist in understanding physical processes. The modelling approach allows the processes to be modelled together, allowing a faster understanding of process interfaces. Note that all aspects are functional, without explicit modelling of physical elements or interfaces.

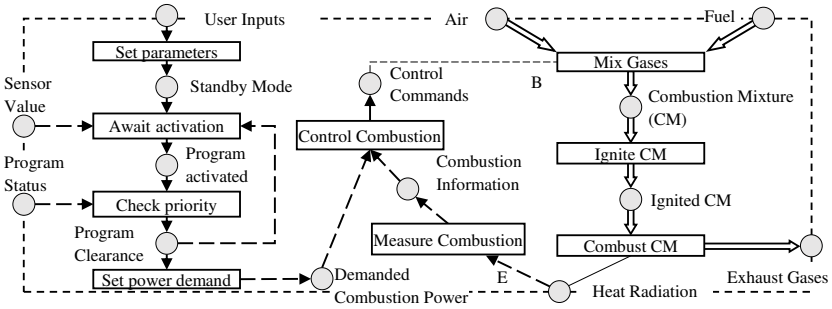

Figure 18: Flow Oriented Functional Modeling of the Combustion and the User Interface

Figure 4 shows the functional architecture derived from the behavioral model in Figure 3, where each cell that contains an “X” is a link between two functions of Figure 3. The matrix allows for compact overview and an integration into database entries, which is important as that data set grows in size.

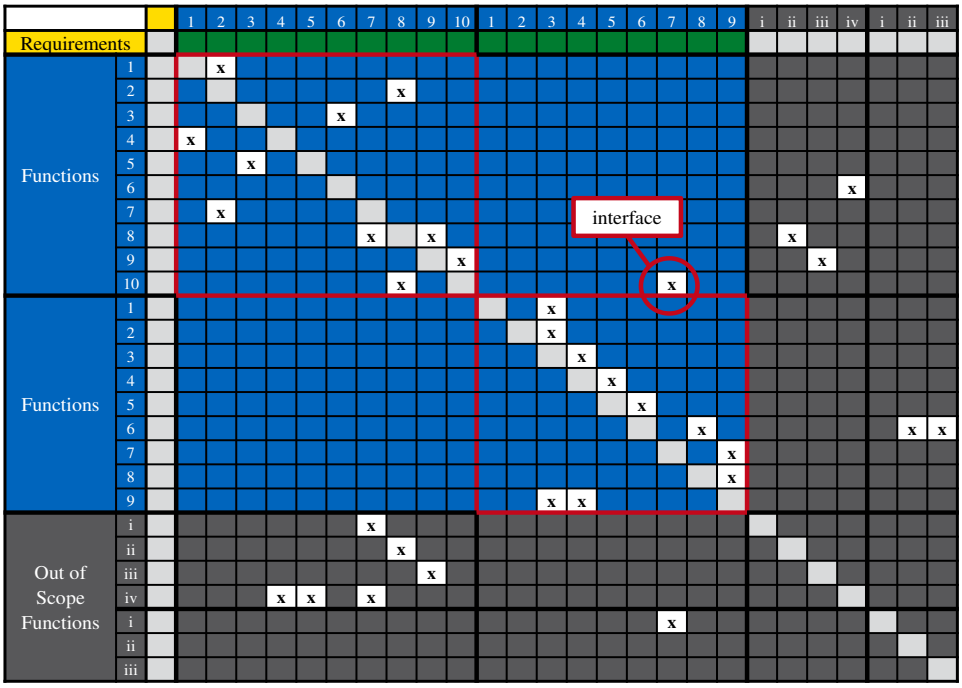

Figure 19: Functional architecture

The two processes, user interaction and combustion, are marked with red boxes. The red circle in Figure 4 marks the single interface in the middle of Figure 3. It shows how this approach helps in managing interfaces in complex systems. In the same way, it can support delegation of authority and responsibility.

The green area at the top of the MDM marks the mapping of non-functional requirements to functions, the architecture decisions. In the specific example, requirements were available only in text form without enumeration. This way it is not possible to clearly separate requirements and assign them to functions. The gray area marks functions that are beyond the product scope. With the whole product contained in the scope, these would be for example functions of the user in interacting with the product.

The case study focused the internal functionality of the heating system regarding the user interaction and the combustion process. The case study contained five functions, two of which are marked in Figure 4. All functions were completed and deployed at the time of the case study. Data was collected through internal documentation and interviews with developers and managers at the industry partner.

6 Conclusion

6.1 Discussion and Limitations

The blueprint presented in this paper offers a flexible approach, adaptable to various levels of available resources. Parts of the blueprint were implemented at the industry partner, verifying the applicability of the implemented aspects. However, it was not possible to determine the applicability of the three control mechanisms presented in the blueprint. A complete implementation of the blueprint was not possible due to time constraints. The results of this research form the basis for the definition of the architecture process at the industry partner, which is currently happening. The results are also used in the structuring of the PLM system of the industrial partner. The industrial partner highlights benefits regarding the control mechanisms of the blueprint and the ability to trace requirements impacting architectural aspects by using the green area shown in Figure 4.

The blueprint itself has limitations regarding the speed of implementation at a company. After determining potential gaps in the existing process and deciding on appropriate methods, it can take years for a company to document the existing architecture, while new products with the old architecture are still being developed. An approach can be to strictly focus on first documenting and modeling the top-level architecture before expanding the level of detail. That way, new products can benefit from early results. At the same time, it leaves the work of detailing to product development, which itself can result in rework cycles and increased development time. However, early results from the architectural process already proved to be a valuable support for discussing interfaces among other things.

6.2 Outlook

To verify the applicability of the entire blueprint and the control mechanisms, an implementation on a product scale is necessary. For this, software tools must be identified that fulfill the modelling needs and which are suitable for a product development environment. Questions about the distribution of responsibility across roles such as system architect and product manager remain to be answered in future projects. Implementations with different amounts of resources in terms of capacity and money would give insight into the scalability of the framework. Using the DSMs and e.g. triangulation for decreasing the required modeling time and optimizing architectures is another field of interest.

Beyond that, further investigating desirable methods for modeling, especially in terms of behavior and architecture is a field of interest. This may take into account novel approaches such as the Elephant Specification Language by Wilschut (2018). The ability for these models to serve the knowledge management of a company through generations of products and architectures is yet to be determined.

7. Acknowledgements

The authors thank the Viessmann Werke Allendorf GmbH for its support during the research project.

References

Blessing, L.T.M., Chakrabarti, A., 2009. DRM, a Design Research Methodology. Springer London, London, 410 pp.

Crawley, E., Cameron, B., Selva, D., 2016. System architecture: Strategy and product development for complex systems. Pearson, Boston, xiii, 465 pages ;

Erden, M.S., Komoto, H., van Beek, T.J., D'Amelio, V., Echavarria, E., Tomiyama, T., 2008. A review of function modeling: Approaches and applications. AIEDAM 22 (2), 147–169. https://doi.org/10.1017/S0890060408000103.

Fuchs, C., Golenhofen, F., 2019. Mastering Disruption and Innovation in Product Management. Springer International Publishing, Cham, 298 pp.

Galle, P., 2009. The ontology of Gero's FBS model of designing. Design Studies 30 (4), 321–339. https://doi.org/10.1016/j.destud.2009.02.002.

Gero, J.S., 1990. Design Prototypes: A knowledge representation schema for design, in: Association for the Advancement of Artificial Intelligence (Ed.), AI Magazine. Winter, vol. 4, pp. 26–36.

Lamm, J.G., Weilkiens, T., 2010. Functional Architectures in SysML, in: Maurer, M., Schulze, S.-O. (Eds.), Tag des Systems-Engineering: München, Freising, 10. - 12. November 2010, 1st ed. Hanser, München, pp. 109–118.

Lindemann, U., 2009. Methodische Entwicklung technischer Produkte: Methoden flexibel und situationsgerecht anwenden, 3rd ed., 354 pp.

Lindemann, U., 2016. Handbuch Produktentwicklung. Carl Hanser Verlag GmbH & Co. KG, München.

Luzeaux, D., Wippler, J.-L., 2017. Beyond the Crawley's FCF, a new paradigm for architecture elaboration and conceptualization, in: 2017 IEEE International Systems Engineering Symposium (ISSE). 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, Austria. October, 11-13, 2017. IEEE, pp. 1–7.

Marti, M., 2007. Complexity Management, 1st ed. DUV Deutscher Universitäts-Verlag, s.l., 277 pp.

Maurer, M., 2007. Structural awareness in complex product design, Online-Ressource (IV, 249 S.).

Schuh, G., Arnoscht, J., Nußbaum, C., 2007. Produktarchitekturen richtig gestalten: Ein Weg zum variantenoptimierten Produktprogramm, in: Industrie-Management: Zeitschrift für industrielle Geschäftsprozesse, vol. 23. GITO Verlag, pp. 29–32.

Ulrich, K., 1995. The role of product architecture in the manufacturing firm. Research Policy 24 (3), 419–440. https://doi.org/10.1016/0048-7333(94)00775-3.

VDI, 1993. VDI-2221: Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte. Beuth Verlag GmbH, Berlin, 44 pp.

Walden, D.D., Roedler, G.J., Forsberg, K., Hamelin, R.D., Shortell, T.M., Kaffenberger, R. (Eds.), 2017. INCOSE Systems Engineering Handbuch: Ein Leitfaden für Systemlebenszyklus-Prozesse und -Aktivitäten : INCOSE-TP-2003-002-04 2015. GfSE e.V, München, 456 pp.

Weilkiens, T., Lamm, J.G., Roth, S., Walker, M., 2015. Model-Based System Architecture. John Wiley & Sons, Inc, Hoboken, NJ, USA, xix, 373 pages ;

Wilschut, T., 2018. System specification and design structuring methods for a lock product platform. Dissertation. Eindhoven, 195 pp.

Wise, R., Sheppard, L., Huggins, J., Broadwell, D., 2015. A methodology and heuristics for re-architecting a legacy system, in: 2015 Annual IEEE Systems Conference (SysCon) Proceedings. 2015 9th Annual IEEE International Systems Conference (SysCon), Vancouver, BC, Canada. 2015. IEEE, pp. 382–389.

Contact: Benjamin Lender, Laboratory for Machine Tools and Production Engineering WZL of the RWTH Aachen University, Campus-Boulevard 30, 52074 Aachen, +49 241 80 28205, b.lender@wzl.rwth-aachen.de