REGULAR PAPER

Feature-driven systems engineering procedure for standardized product-line development

Christian Granrath M.Sc.1,2 | Christopher Kugler M.Sc.1 | Sebastian Silberg B.Sc.1 | Max-Arno Meyer M.Sc.2 | Dr.-Ing. Philipp Orth1 | Dr.-Ing. Dipl.Wirt-Ing. Johannes Richenhagen1 | Prof. Dr.-Ing. Jakob Andert2

1 FEV Europe GmbH, Aachen, Nordrhein-Westfalen, Germany
2 Teaching and Research Area Mechatronics in Mobile Propulsion, RWTH Aachen University, Aachen, Nordrhein-Westfalen, Germany

Correspondence
Christian Granrath, FEV Europe GmbH, Aachen, Nordrhein-Westfalen, Germany. Email: granrath@fev.com

[Correction added on 22 January 2022, after first publication: The affiliation of the first author has been updated.]

Abstract
Numerous systems engineering (SE) methods for the model-based and textual specification of systems focus on managing complexity solely by partitioning the system based on physical structures or by defining different views of the system and therefore reach their limits in agile development. The increasing demand for an agile system development requires an agile systems engineering procedure for the model-based and textual top-down specification of systems. Although function-based development, variant management, and product line development are well established in software engineering, previous work has failed to introduce methods for the agile specification of systems by combining established methods from systems and software engineering. For that purpose, this paper demonstrates a new SE methodology, which for the first time combines conventional SE methods with the agile development procedure of feature-driven development. The methodology is systematically developed based on theoretical analyses and its suitability for the application-specific definition of feature-driven development processes is demonstrated using the example of reference architectures for XIL simulation models of electric vehicles. By applying feature-driven development, the resulting CUBE methodology enhances collaboration in interdisciplinary development teams and enables companies to adapt development processes to a more agile top-down specification of systems.

KEYWORDS
Architectural Design, Model-based Systems Engineering, Processes, Requirements Elicitation and Management, Systems Thinking
1 | INTRODUCTION

In the last decades, the increasing complexity of products in various industrial sectors has led to a growing importance of systems engineering in conjunction with project management.\(^1\)\(^,\)\(^2\) Especially in the automotive industry, component-based development of the system was established, which resulted in a division of responsibilities and a definition of the system elements according to the physical separation of the system or the development disciplines involved. The insufficient coordination between the disciplines involved and consideration of interdependencies on the upper hierarchical levels of the system resulting from a lack of system thinking led to low specification quality in the form of incorrect, inconsistent, or incomplete requirements.\(^3\)\(^,\)\(^4\)

For relatively simple systems where much experience is available or where the innovation rate is low, engineers were able to cope with this while bypassing the official process. Often this led to problems with the integration of the system elements to a complete product, caused by incompatibilities and gaps between the requirement specification and the test cases, finally leading to a rise in development costs and time.\(^4\)\(^,\)\(^5\) The trend of increasing system complexity, caused by increasing networking (e.g., in the fields of Advanced Driver Assistance Systems, Automotive Connectivity,\(^6\) or Internet of Things,\(^7\)) or increasing degrees of automation, and at the same time shortening development cycles, additionally increase the cost pressure for manufacturers and require high specification quality to avoid delays in the development process.\(^3\)\(^–\)\(^8\)\(^\)\(^–\)\(^9\)

One core strategy for solving the conflicts between time-to-market, cost reduction, and emerging quality problems is the frontloading of development activities.\(^4\)\(^,\)\(^9\) This requires increased effort in early development stages, such as requirement specification and architectural design.\(^9\)\(^,\)\(^10\) One possible type of frontloading is model-based systems engineering, which focuses on the system-based consideration of products and associated processes in all phases of the product life cycle.\(^4\)\(^,\)\(^8\)\(^,\)\(^11\) A special meaning is assigned to the system-based development of architectures and requirements on basis of continuously extended models.\(^3\)\(^,\)\(^12\)\(^–\)\(^14\) Model-based systems engineering requires high coordination efforts between all relevant stakeholders in the early phases of the development, but allows a complete consideration of relevant system interdependencies and technical and subject-specific aspects, which leads to an increased quality of requirements.\(^3\)\(^,\)\(^4\)

In contrast to classical mechanical engineering, system-based product development is widely established in the area of software, taking into account solution-neutral and product-specific views. Methods for the suitable reuse of system elements and for the realization of variant management have become indispensable in the field of software product line development, but are only rarely used in the design of hardware-based systems.\(^7\) Model-based specification can contribute to improve communication in interdisciplinary development teams, to the avoidance of inconsistencies between requirements as well as errors and failures that would otherwise only be perceived at later verification levels.\(^4\)\(^,\)\(^8\) A transfer of these methods for the design of hardware-oriented products or embedded systems represents a large optimization potential in mechanical engineering dominated industries.

Additionally, agile development is continuously gaining relevance also in companies dominated by mechanical engineering.\(^7\)\(^,\)\(^15\) The shortening of development cycles enables an improved cooperation, a faster reaction to changed boundary conditions and thus an increased quality of the development artifacts.\(^7\)\(^,\)\(^16\)\(^,\)\(^17\) The application of development procedures, which have already been validated and established in the field of software engineering, can further contribute to the optimization of product development.

The demand for an agile system specification and development as well as the increase of development efficiency by using established methods of software development leads to the following objective: Definition of a system-based and feature-driven procedure for the model-based and textual top-down specification of systems.

For this purpose, this paper systematically presents a new SE methodology based on theoretical analyses. This combines for the first time SE procedures with the agile development procedure of feature-driven development to enable the standardized top-down specification of products and product lines. Chapter II starts with an overview of relevant technical terms in the field of systems engineering and the state-of-the-art of SE methods, processes, and procedures. In chapter III.A, all objectives to be addressed by the underlying procedure model of the new methodology are elaborated as well as a detailed description of the procedure model (chapter III.B) is performed. The following step-wise explanation of the individual phases is combined with the presentation of a rough example (chapter III.C). In the next step, the way for considering functional system variants is shown (chapter III.D). By using the example of standardized simulation architectures of electrical vehicles, the suitability of the procedure model for the application-specific definition of feature-driven development processes is demonstrated (chapter III.E). The definition of agile or conventional specification processes on the basis of the presented procedure is explained in chapter III.F for industrial purposes. In the last two chapters (chapter IV and V) all presented contents are critically reflected and summarized.

2 | BOUNDARY CONDITIONS FOR METHODOLOGY

The first section of this chapter is intended to ensure a consistent interpretation of technical terms in the field of systems engineering and a common understanding of the presented contents. Following, an overview of existing methods and procedures as well as the underlying targets in the area of SE will be presented.

2.1 Technical terms in systems engineering

A system is a collection of elements and a collection of relationships between these elements so that they can be regarded as a limited whole in relation to the elements surrounding them.\(^16\) The term element is used in the broadest sense to capture simple physical
things, complex organisms (including humans), environments, and technologies, as well as organizations of humans, information, or ideas.\textsuperscript{18} Open systems exist in an environment described by related systems with which they can interact and under conditions to which they can respond.\textsuperscript{18}

Systems engineering is intended as an “interdisciplinary approach” to enable the realization of successful (engineered) systems.\textsuperscript{11,18,19} It integrates all disciplines and specialty groups into one team and forms a structured development that takes into account all phases of the system life cycle.\textsuperscript{11,19} One of the main tasks in systems engineering is to work with stakeholders (including customers), represent their points of view to the team and the team’s perspective to them, and create a constructive environment for teams, including bridging cultural and communicative differences in multidisciplinary teams.\textsuperscript{11} In this context, unambiguous definitions of all technical terms are of crucial importance in order to achieve unequivocal and clear communication.

In recent years, this problem has already been addressed in numerous publications and standards in the fields of systems and software engineering. However, the analysis of several publications shows an inconsistent use or an unestablished, non-uniform understanding of the technical vocabulary in the field of SE. In order to clarify the meaning of these terms in context of this publication, the central and relevant terms are defined by extending and summarizing existing publications and standards. It is important to distinguish the meaning of the terms methodology and method, process and procedure as well as activity and approach:

“A methodology can be defined as the collection of related activities, methods, procedures, processes and tools used to support a specific discipline or a class of problems that all have something in common.”\textsuperscript{11,12,20}

In other words, a methodology is essentially a “recipe” to address a common group of problems.\textsuperscript{12,21} No detailed application directives are made, for example, standardized duration of phases of the procedure, but only means are provided for addressing the problem or the group of problems, which must be adapted project- and product-specifically.

The associated processes represent a group of elements of a methodology (cf. Figure 1). These include the definition of activities for the realization of a desired result on the basis of provided input information. In addition, all time aspects, such as the duration of activities or necessary start/end points including the necessary resources, their roles and responsibilities as well as the resulting work products are defined.

“A process is a set of interrelated activities that is concerned with transformation of input to output, including definition of roles, responsibilities, milestones, work products, duration, resources and not concerning how the required output is obtained.”\textsuperscript{8,12,20,22–24}

Activities represent the sub-elements of a process.\textsuperscript{25} In turn, they can be broken down into tasks and are characterized by time expendi-

![Figure 1: Relation of technical terms in systems engineering](image-url)

“An activity is a set of distinct scheduled tasks that consume time and resource and that are assigned to perform the realization of necessary outcomes.”\textsuperscript{46,20,25–27}

When defining processes or activities, no concrete description is given of the way in which a process step is to be performed. Likewise, no logical dependencies of steps are defined. These contents are represented by methods, which can be used for the execution of the process steps.\textsuperscript{12,28,29} Exemplary is the specific transfer and application of model-based development for the execution of an activity in the development process.

“A method is a systematic way to accomplishing projects or solve problems through detailed and logical plans based on a specific mindset.”\textsuperscript{12,28–31}

Established methods that specify concrete sequences of steps for the implementation of activities or processes and whose application achieve reliable and consistent results are defined as procedures. The waterfall model, V-model, or similar approaches are exemplary models.
for the presentation of an established procedure. In contrast to the process, procedures only define logical dependencies of steps during execution, but without defining time sequences, durations, responsibilities.

A procedure is an established method of accomplishing a consistent performance or result. A procedure typically defines the sequence or ordered series of steps to perform an activity or process.

In the field of systems engineering, the term approach is often used and inconsistently equated with the meaning of the previously defined terms. The currently valid versions of the norms relevant to SE do not define the term approach. In common parlance, an approach is understood as a specific way or idea to deal with a problem. For this reason, it will be used in the following as a synonym for the term method.

### 2.2 State-of-the-art systems engineering processes and procedures

In the field of systems engineering, several procedures and methods have been published in recent decades. In the following, single publications will be exemplarily presented. An exhaustive description of all existing methods is expressly not guaranteed.

One of the best known standards in the field of SE is part of ISO Standard 15288—Systems and Software Engineering—System life cycle processes. The standard provides a “set of related processes” that can be used to design appropriate system life cycle models appropriate to their products and services. The provided descriptions are divided into four groups (agreement processes, technical management processes, technical processes and organizational project-enabling processes). The philosophy behind this standard is based on the ability of the system engineer to divide the presented descriptions into an order of activities that are applicable to the program. At the same time, there is no specific logical order presented for sequencing activities, nor are durations, resources, or tools defined. For this reason, it should be understood not as processes but as a set of activities necessary to achieve a desired result.

A similar objective is pursued by the BWI Hall Concept. This method places particular emphasis on general applicability, is not oriented to specific fields and is intended to offer support where other SE methods do not provide an answer. The method is to be regarded as a formal framework that provides methods without prescribing a procedure. For this purpose, general descriptions of the “SE philosophy,” for example, system thinking, as well as four essential modules for the design of systems are provided. The first module is called “Top-Down” and describes the continuous decomposition of the whole system into smaller elements whereby the detailing is increased (cf. A IV). The second model “Variant Development” contains the formation and consideration of possible variants in order to consider all possible solutions of the system. The “phased approach” (project phases) as the third module represents a planning instrument (macro instrument) for deciding which activities are necessary to achieve the desired results in different project phases (management-oriented components of the model) (cf. A IV). The problem-solving cycle is the fourth module and should be used in the implementation of all phases, corresponding to the third module. It forms the micro-strategy for structuring the phase cycle. The situation-related combination of the building blocks is intentionally left open, so that the users themselves should choose which and in which form, modules are used. A transfer of the method to agile applications is made possible by the modularity and thus good adaptability to the development processes.

According to the definitions presented in the last section, the “Systems Engineering Method” is a scientific procedure to the engineering of a complex system. Four steps are defined, which are to be run through systematically in order to determine an optimal solution based on customer needs (cf. Figure 2). At the beginning, all requirements to be considered (requirements analysis) are collected, which are then used for functional descriptions of the system (functional definition). The third step involves the synthesis of a series of alternative system components representing a variety of design solutions to implementing the required functions (physical definition). Systematic, physically oriented partitioning is recommended for system design, but no clear instructions are given for implementation. The final step is the validation of the previously defined solution (design validation).

In each of the steps mentioned, knowledge could be gained that would make it necessary to repeat the previous step. In particular, the validation can lead to an additional specification of new requirements that were not considered at the beginning, which requires a complete, iterative execution of the steps. The described four-stage procedure can be applied in different project phases. Depending on the project phase, the focus of the development is on different aspects, so that the scope of the activities must be adapted.

Summarized, both the BWI Hall Concept and The SE-Method focus on the development of the optimum solution for a system and not for a product line, taking different variants into account. The definition of variants is used here decisively to define an optimized solution by selecting from a number of different, possible solutions. An iterative application of The SE-Method, similar to the BWI Hall Concept, is given by the general applicability.
The Object-Oriented Systems Engineering Method (OOSEM) is again a stepwise procedure for the design and specification of systems (cf. Figure 3). Analogous to The SE-Method, the first step is to analyze the needs of the stakeholders. In a second step, the system requirements are analyzed, followed by the definition of a logical architecture. In the last step, possible physical system architectures are considered. Overall, OOSEM represents a collection of activities for each phase of the quadrinomial procedure that can be used for specification of systems. Procedures for the implementation of the individual steps must be defined application-specific by the user. Special focus is put on the model-based implementation of the specification, which is realized by an initial creation of a system model. Furthermore, parallel activities for variant evaluation and traceability of requirements are demanded.

Analogous to the methods and procedures presented so far, the definition of the system elements at OOSEM is carried out based on the physical system structure. A feature-based development is not explicitly considered by OOSEM, BWI Hall Concept and The Systems Engineering Method, even if the functionality is described on a system level and partitioned into functional elements. An explicit description of the hierarchy levels to be considered is not given by OOSEM. The definition of logical architectures supports in particular the consideration of variation points as well as the associated possibility to specify product lines. Consequently, this procedure does not focus exclusively on the specification of an ideal solution for the realization of the requirements.

The lesser-known SE methods SPES (Software Platform Embedded Systems) (cf. A II) and its extension SPES XT as well as SMArDT (Specification Method for Architecture, Design and Test) (cf. A I) and its project-specific extension MTSF (Model-based Testing of Software-based Functions) focus on the system-based design of embedded systems. These methods combine four viewpoints respectively abstraction levels with different granularity layers respectively decomposition levels. Analogous to OOSEM, the four viewpoints are divided into consideration of all relevant requirements (requirements viewpoint), description of functions assigned to the system or system elements (functional viewpoint), definition of logical architectures (logical viewpoint) as well as of the technical system realization (technical viewpoint). The logical system view is not emphasized by SMArDT and MTSF, but the “realization” is defined as the fourth system view. After defining the technical solution, the details specific to the implementation are specified and then implemented. The granularity levels of SPES and SPES XT correspond to the physical partitioning of the system, SMArDT and MTSF also pursue additionally the goal of function-based development. For this purpose, the four abstraction levels are used to describe a system from the point of view of a feature. A clear separation of the function-based consideration and the physical partitioning does not occur.

The aspect of variability is made possible in the methods mentioned, analogous to OOSEM, by the consideration of a logical architecture. Furthermore, SPES XT defines variability as orthogonal relationship, which has to be considered in the system design. Concrete procedures, how the specification of product lines is to be realized, are not described in detail. Additionally, SMArDT and MTSF focus in particular on the aspect of verification and validation of the specified system. The model-based specification is used for the automated generation of test cases by means of high formalization of the artifacts. The increased efforts for the creation of model-based and formalized artifacts are more than compensated by strongly reduced efforts for the test case definition. The application of SPES XT and MTSF for agile development processes as well as the consideration of temporal product versions are not explicitly defined by the methods.

The procedures and methods described in this section target the improvement in all phases of the development cycle. They differ in the addressed life cycle phases, the intended application contexts and the focused aspects of development. Depending on the planned field of application, project or system to be developed, the advantages and disadvantages of the methods must be weighed up and the most suitable must be selected accordingly.

The need to implement systems engineering in a more agile way has led to the derivation of key characteristics of agile development for the field of SE. Based on the assumption that agile is much more a mindset than a set of rules to be followed without regard to their appropriateness, the consideration of these characteristics should enable more agile development based on unchanging SE fundamentals. In this way, the customer should prioritize the requirements, progress should be measured against operational features, and lessons learned should be used to adapt both processes and the system under development to meet the customer’s needs. Short iterations with the goal of executable work products, continuous integration, lean processes that only specify the essentials, and a focus on team ownership should also contribute. As an example, Harmony aMBSE can be cited, which defines a concrete procedure for the development of optimized systems with a focus on agile key characteristics. For system design, a model-based procedure is proposed using different system views and physical partitioning. The realization of incremental and agile system
development is made possible in particular by the use of numerous iterations during development.\textsuperscript{40} The assumption that systems are becoming larger, more complex, and more software-based again reinforces the use of agile development. By focusing on software development, SE can leverage the flexibility and adaptability of software to define the system and its components in such a way that software development is less complex and the system architecture and design supports the effectiveness of software development.\textsuperscript{39} In particular, a common high-level system architecture can enable both the process and the product to respond effectively to new and immediate situational requirements.\textsuperscript{41}

A combination of the agile approach of FDD and at the same time a physically oriented decomposition of the system is not yet addressed by the existing methods and procedures, can contribute to the implementation of the agile key characteristics and will therefore be investigated in more detail in the following.

\section{METHODOLOGY}\label{sec:methodology}

The following chapter presents the novel systems engineering methodology - Compositional Unified System-Based Engineering (CUBE). CUBE can be seen as an extension of SMA\textsuperscript{4} and combines established methods from systems and software engineering, such as the agile procedure feature-driven development.\textsuperscript{11}

In the field of SE, the use of the System Modeling Language (SysML) is well-established for model-based system specifications. The language provides a large number of structure and behavior diagrams to define systems from different perspectives and at different abstraction levels.\textsuperscript{4} However, it does not provide detailed instructions as to which diagram types to use in which order or for which abstraction level.\textsuperscript{3,38} A textual specification is possible as a supplement to or exclusive to model artifacts. The new methodology shall enable textual as well as model-based specification of products with different kind of software tools suitable in this context. Due to the large number of established software tools for textual and model-based specifications, the aspect of tool support will not be dealt with in the chapter III-A-D, but instead the focus will be on the procedure model underlying the methodology. In section A, the underlying objectives are presented. Sections B-D focus a description of CUBE and the covered aspects as well as a description of the theoretical, ideal sequence of phases for the system-based top-down specification of products and product lines using the new procedure. In chapters E and F an application example as well as explanations for industrial application to realize agile processes are presented.

\subsection{Objectives}\label{sec:objectives}

The following procedure focuses and is at the same time limited to the system-based and feature-driven specification of products, whereby the product to be specified can be understood as a system according to the definition in section II.A. Consequently, in the specification of the system, its elements and the internal and external interactions, the concept and development phase in the system life cycle are addressed.\textsuperscript{11} Consideration of the subsequent development phases, such as production, utilization, and retirement, is not carried out.\textsuperscript{11} Furthermore, the procedure exclusively describes the core area of development. Support processes, such as supplier management or product management, are not described in detail.

The conceptual design of a product is characterized by different trends, such as rising complexity due to the increasing connectivity and intelligence of systems, from which challenges result that must be addressed in an appropriate manner.\textsuperscript{5,6,8} In the following sections the objectives are presented, which address current challenges in product development and have led to the development of the new procedure model.

According to the principles of agile development, customer satisfaction is the most important asset in product development and the most decisive factor for the success of a product.\textsuperscript{17} If the customer is not satisfied with the product, the product is not bought and recommended, which leads to a decline in sales figures and thus to low long-term sales for the manufacturing company. Therefore the focus of the entire development must be on the satisfaction of the customer. Consequently, all the wishes and needs of the customer must be taken into account when developing the product (analogous to the OOSEM procedure, The Systems Engineering Method and SMA\textsuperscript{DT}). The analysis and specification of these needs must be done in cooperation with the customer to determine the actual needs, which often cannot be fully specified by the customer itself at the beginning of the development. The procedure must support this analysis and specification by defining required activities. In this context, the customer can not only be understood as the end customer who acquires the product, but also internal stakeholder that has to be considered. Furthermore, the interactions of the product with other systems must be considered in order to ensure correct operation in all use cases. This is the main objective in the development of a product:

Objective I: Specification of the product or product line with focus on stakeholder needs, respectively their expectations, as well as the system context

The creation of specification artifacts during the system design phase is very time-consuming. Too much effort drives up development costs, which can result in a lower company profit. Therefore, the ideal procedure has to limit the number of artifacts required for specification as well as offer measures for controlling product development complexity. A limited number of specification artifacts prevents excessive coordination and understanding efforts as well as redundant or identical work results within the documents. At the same time, too few artifacts lead to increasing complexity of each artifact. For this reason, a continuous extension of specification artifacts is targeted, which reduces the number artifacts compared to the creation of individual ones in each development step. The definition of different views on the system allows a focused analysis and can be used to structure the artifacts.\textsuperscript{3} The methods and procedures mentioned in the previous chapter provide system views to be considered in this context. By using a database to create model-based artifacts, analogous to OOSEM, an increased consistency of the artifacts is additionally guaranteed.
Objective II: Continuous extension of specification artifacts to ensure fulfillment of stakeholder expectations by definition of various views on system elements

The development of products, as described in Chapter I, is nowadays characterized by increasing system electrification, networking and communication of systems (internet of things), high and very specific customer demands, a large variety of variants and increasing legal requirements for compliance with ecological goals. These boundary conditions result in increasing system complexity. Partitioning or decomposition of the system according to the physical, technical structure is possible as a measure to control complexity (analogous to OOSEM, SPES XT and The SE-Method). The complex system is divided hierarchically into small, manageable system elements in order to make complex interrelationships within the considered system boundaries traceable.

Objective III: Layered decomposition of the system, considering clear definitions of system boundaries for all decomposition elements, to reduce complexity of the specification elements

A further possibility to control increasing system complexity is the function-based or feature-driven development of the system. Instead of a development oriented to a physical system structure, the system is described from the point of view of a set of functions that each produces a desired result (by the customer). This consideration is independent of the technical implementation of the operating principle within the system and can be realized by one or more technical subsystems. The function-based specification represents an additional, alternative and focused method, which in particular enables a solution-neutral view of the product and hence supports agile development in short cycles. In order to realize the solution-neutral view, a clear differentiation to the physical decomposition has to be ensured.

Objective IV: Feature-based specification of the product to enable solution agnostic top-down development

One of the greatest challenges in SE today is the very high number of product variants. The definition of suitable measures for handling the increasing system complexity caused by the high number of variants is an essential influence on the costs in the development process and decisive for the company profit and success. In the field of software development, the term “software product line development” is used to describe the development of a reusable platform for a specific domain, which makes it possible to efficiently derive various adapted products and thus significantly reduce development costs. The analysis and definition of variant-independent and thus reusable development artifacts is possible by the analysis of commonalities of the products and at the same time definition of variation points, for example, in the form of feature models. The avoidance of repeated and redundant development activities as well as the rejection of results of intensive analyses achieved in this way can significantly contribute to the reduction of development costs and risks. The increasing cost pressure in product development requires the application of analogous methods for all system elements of the product, regardless of whether it exclusively contains software. Especially the application in early development phases can already support the reuse of specification artifacts. OOSEM and SPES XT deliver first methods for the specification of product lines by the definition and specification of a solution-neutral view of the system.\footnote{11,37}

Objective V: Consideration of system variants of a product or a product line, including variation points on different decomposition levels, to improve reusability of specification artifacts

In addition to the product differences, the companies also differ greatly with regard to the development processes used. Different corporate cultures, development environments as well as the mind-set are influenced by the industry, the product and the company size and result in diversified development processes. The necessity of a continuous and fast adaptation to requirements unknown at the beginning of the project and constantly changing in the course of the development mainly justifies the trend towards agile development in recent years. This has contributed once again to additional diversification. For these reasons, a flexible and scalable adaptation of the specification procedure to companies with conventional or agile development processes and according to the size of the company or the development team is desired.\footnote{47} In this context iterative adaptation development (analogous to the BWI Hall Concept or OOSEM) or the application of agile methods can be considered.

Objective VI: Procedure as base for definition of conventional and flexible, agile specification processes as well as for consideration of functional product versions

3.2 Overview of compositional unified system-based engineering

Compositional Unified system-Based Engineering (CUBE) is targeting generic, system-based specification of products and product lines based on stakeholder needs by providing a standardized, uniform procedure for the specification of systems, regardless of the industry. The CUBE model (cf. Figure 4) illustrates five central dimensions that are recommended to be taken into account when specifying a system and which are covered by the CUBE procedure.

As an extension of MTSF and SPES XT, the compositional character is a central dimension of the procedure (addressing Objective III). As a base for development, the system context of the product to be developed (System of Interest) is analyzed and the related stakeholders and other interacting systems are determined. The network of interacting systems, also called System of Systems (SoS), in which the System of Interest (SoI) is embedded, represents the highest decomposition level to be considered, since this consideration determines the interfaces of the SoI to the system context as well as the requirements for the SoI by stakeholders and other autonomous and interacting systems.
On the second decomposition level, the considered system boundaries with respect to the specification focus are reduced to the SoI. The third decomposition level describes the first partitioning of the SoI according to the physical/technical elements of the SoI to define clear system boundaries of the specification elements and complexity reduction through limited system size and focused considerations. The decomposition can be continued in an arbitrary number of steps, whereby each element of the outgoing plane can be decomposed into an arbitrary number of elements.8,11

The second central dimension is the definition of five different abstraction levels to represent different views of the system with a consistent system boundary (addressing Objective II). An abstraction level, which is independent of the physical decomposition of the system, allows a focused view on a system element, independent of the decomposition layer. Furthermore, it enables a continuous extension of the specification artifacts for the focused decomposition element with stepwise addition of more detailed system information.

The first level of abstraction focuses on a black box view of the focused element and includes the specification of all external requirements for the system, such as stakeholder needs (customer value) and expectations of other involved systems or actors.11 Furthermore, the interfaces to these adjacent systems and the relevant system context are defined. This view on the considered element describes the most abstract view on the system. The following levels of abstraction describe a white-box view of the system. The main focus here is on the functionality and the internal structure of the system.11 The second level of abstraction (operating principle) describes a solution-neutral view of the system, whereby the operating principle and the functional behavior of the system are specified without considering technical implementation alternatives or structural/architectural decisions. The latter aspects are part of the consideration of the third level of abstraction (technical solution), which in turn is divided into two layers. The first of these layers focuses on the specification of a logical system architecture (logical architecture). The functional elements, as part of the previously defined operating principle, are logically grouped by structural elements and are thus for the first time subject to a structural realization decision in the form of the logical product architecture. Nevertheless, at this level, no technical decision is taken on the realization itself, but only functional logical groups are defined. The background lies in the specification of product lines and at the same time poses a measure for variant management as well as suitable reuse of specification artifacts (addressing Objective VI). The specification artifacts of the logical groups represent reusable elements of the product line that do not contain product-specific characteristics and are therefore valid for all possible product variants of the product line. This form of architecture thus defines the core of the product line, which remains the same and is used in all product derivatives, in the form of functional-logical, multi-product architectural elements, as well as the technical variation points of the product line. In the fourth level of abstraction (technical architecture), these above-mentioned elements are extended by technical implementation decisions. Consequently, all required decisions for the development of a concrete product of the product line are made. On the basis of the reusable, logical architectural elements, the technical architecture is defined by adding product-specific information or by determining a final configuration of the product. A distinction between logical architecture and technical architecture is not generating added value in the case of the specification of an individual and already existing product (reverse engineering), which is not to be part of a product line in the future and does not change in time, since in this case all logical elements exist only in a time-independent and technically unequivocally defined form. The last and most detailed level of abstraction (realization) contains the specification of all implementation and production specific aspects of the technical architecture to be realized. At the end of this phase, the specification for the considered decomposition element is fully completed and, as a subsequent phase in the development process, forms the basis for implementation.

The third relevant dimension illustrated by the model is the consideration of different development phases during specification
(addressing Objective II). Thus it is not sufficient if in the concept and development phase only requirements are created for the design of the system. The consideration and specification of at least one test case for each requirement must be ensured already during system design in order to guarantee a successful implementation of the functionality required by the product. For this reason, the model also highlights crucial elements of the verification and validation phase, which must already be considered in the specification phase. In this context a manual creation of test cases but also an automatic test case generation, analogous to MTSF, can be applied. The application of test case generation requires the creation of a consistent and formalized system model (according to MTSF and OOSEM), which can be used as a basis for test case generation.

As a fourth aspect, the specification of functional product variants is addressed (addressing Objective V). In contrast to the technical variants mentioned in the paragraph on abstraction layers, these do not diverge through functionally identical implementation decisions, but rather exhibit different functionality. Thus, the system to be developed can be realized by different operating principles and still fulfill the same customer benefit equally, for example, different types of powertrains in a vehicle to fulfill the customers demand for mobility. In comparison, the choice of the product design (e.g., the product color) represents a non-functional aspect, would require an identical functionality and for this reason would be assigned to the area of technical variability.

As the last dimension, the model illustrates versioning of the product specification (addressing Objective VII). In this context, versioning must be considered from two different perspectives. On one hand, the product itself can have different product versions in terms of time and function due to further development and changing customer requirements. The procedure for specifying the product must therefore offer the possibility of documenting these different product versions. On the other hand the specification of a certain product is subject to a temporal duration, so that during the specification phase, in particular up to the complete conclusion of the specification, a versioning of the artifacts is necessary. Depending on the type and size of the project team or the company, the complexity of the product, the company structure and the development processes, an adaptation to agile and flexible or also conventional development processes is required. The type of process strongly influences the frequency of the necessary versioning of specification artifacts. The procedure model can be used as base for the definition of different specification processes and thus provides a possibility to address these challenges in the context of versioning.

3.3 Product specification procedure and application

In the previous section, the CUBE model with its recommended dimensions was presented. A concrete sequence of steps to pass through the levels of abstraction and decomposition defined in the model and which relates the dimensions has not yet been considered. The application of CUBE for process definition within a development project is always subject to project-, company-, and product-specific constraints. Nevertheless, based on theoretical considerations, there is a recommendable sequence of steps for addressing the objectives mentioned in Chapter III.A, which is defined in the form of a procedure (cf. Figure 5).

The basis for the specification of a system (SoI) is a consideration and definition of the system boundaries of the SoI, its system context and the system stakeholders. On the basis of these clear definitions, the expectations or needs of the stakeholders with regard to the system-of-system as well as the interactions of the SoS with its system context can be specified in the form of black box requirements in a first step. These include both functional as well as non-functional requirements. The relevant stakeholders can be manifold, so that an analysis considering different views on the system, for example, for architectural aspects using "4+1 View Model," is recommended. The specification artifact resulting from the first step (Customer Value Specification of System of Systems = CVS(SoS)) forms the central basis for the specification with focus on stakeholder needs (addressing Objective I). An example of this development step can be the specification of the end customer’s requirements or the statutory boundary conditions for the SoS.

By definition, a SoS is a group of systems characterized by the operational independence of the individual systems and the fulfillment of an individual customer benefit of each individual system. One example is the consideration of a group of interacting and collaborating respectively networked vehicles in the development of an automated driving function. The procedure focuses on the specification of the previously defined SoS at the top decomposition level. Especially if by definition no SoS exists for the considered SoI, the specification of all interactions with the system context or other systems and actors is considered on the first decomposition level and the fulfillment of an individual customer benefit is focused to define the decomposition levels. Thus, the powertrain of a vehicle does not fulfill the condition of operational independence, but fulfills an individual benefit for the overall vehicle manufacturer. In this example, the term SoS is transferred to the vehicle, which contains the powertrain (SoI) as an individual subsystem fulfilling a specific customer benefit. Consequently, the term SoS is extended in the procedure, if there is no SoS according to definition, and is understood as a group of interacting systems, each of which realizes an individual customer benefit for at least one stakeholder. In addition, the SoS contains the SoI to be focused, whereby the SoI is regarded as a black box and the other interacting systems are functionally separated from the SoS.

As a second step of the black box consideration at the highest decomposition level, the definition of functional features follows for addressing the determined customer needs. According to the FDD procedure, this is done in the form of a feature list FL(SoS). Each requirement previously documented in CVS(SoS) must have at least one unique feature assigned to it, whereas a feature can also address multiple requests. This results in a set of features $F_{Sos} = m$, where at least one functional feature must be defined to describe the focused decomposition element ($m \geq 1$). As an example of a feature for the aforementioned vehicle fleet, “Connected and fully automated driving of a vehicle fleet on highways” could be named.
In the following White-Box view of the SoS, the function-based specification is used to describe the focused decomposition element only from the point of view of the focused feature (addressing Objective IV). For each feature defined in the feature list, the solution-neutral operating principle is documented as part of the feature-specific Operating Principle Specification OPS(Fxy). This type of consideration, based on a function-based decomposition, enables a system structure- and realization-independent specification and at the same time a further, suitable possibility for handling high system complexity. In this step, solution-neutral actions are defined for the example feature (e.g., determination of the distance to the front vehicle), which are necessary for the realization of the feature, but which do not carry out a definition of the technical realization (e.g., by means of a radar or lidar sensor).

In the next step, the focus is shifted to the next level of decomposition to focus on the SoI. The other elements defined in the logical architecture of the SoS are not considered separately, since they are not part of the system to be developed. The information addressed to the SoI from the logical architecture specification of the SoS (LAS(SoS)) is used as the starting point for the black box analysis and the accompanying description of the customer values. These are both functional requirements and specifications of interactions with other systems or actors in the form of system interfaces. The assumption of these requirements documented before on the higher levels of decomposition, is of crucial importance for the following functional specification of the Sol. The clear interface definition at a higher hierarchy level bases as starting point for the specification of the Sol and improves the system integration. In this context, information exchanged on the status of vehicles within the fleet, such as the current vehicle acceleration of the lead vehicle, can be mentioned as exemplary input interfaces of the Sol.
With the further addition of customer-specific requirements to Sol that were not part of the consideration at SoS level, the definition of a new group of functional features (e.g., fully automated and energy-efficient transport of vehicle passengers on highways) for addressing the customer requirements addressed to Sol ensues. Analogous to the procedure on the previously specified decomposition level, a feature-specific description of the Sol is used to specify an operating principle and a logical architecture for each feature. The combination of the data from the feature-specific logical architectures is in turn used to define the logical architecture of the Sol. The architectural elements contained represent the structure and configuration of the Sol and are considered separately on the following decomposition level. In contrast to the transition from the first to the second decomposition level, all elements described in the LAS(Sol) must be considered, as this is the only way to ensure a complete specification of the Sol. With the optional addition of specific customer requirements, a group of system element specific features is defined and the feature-based procedure, analogous to the previous decomposition levels, is applied. This can be used to systematically decompose the system using an arbitrary number of hierarchy levels to reduce system complexity.

After completion of the specification of the logical architectures on the system-specific, relevant decomposition levels, a final technical product architecture is completely defined. In this step, the final implementation decisions are made. Each logical architectural element is allocated to a unique realization, such as the decision to execute an element as software or hardware or the decision to use certain hardware components or variants. For the logical group "environmental perception" mentioned in the previous section, a final, technical realization would be determined in this step, such as the realization in the form of a radar sensor, a camera and 2 lidar sensors. The resulting hardware and software architecture is then completely specified, which requires a complete definition of the static and dynamic aspects of the architecture, with non-functional requirements in particular having a high impact. For example, it is necessary to define a unique naming convention, data types, permissible value ranges, sample time, etc. for the explicit specification of interfaces in the software architecture. These decisions can neither be made top-down nor bottom-up over the decomposition levels, since decisions on the highest decomposition level are strongly dependent on decisions on lower decomposition levels and vice versa. For example, the definition of a vehicle powertrain architecture can have an influence on the technical realization of individual components on lower decomposition levels. This dependence between realization decisions requires a temporally parallel definition of the technical architectures.

In the case of the specification of a single product respectively system for which no functional or technical variants exist and which does not change over time or which does not consider the aspect of versioning, a specification of the logical architecture can be skipped for reasons of efficiency. In this case, the realization of the logical architecture elements is defined by a definite, singular relationship, which is why the logical and technical architecture specification would represent redundant information. It is therefore recommended to omit the logical architecture specification. In this special case, the decomposed elements of the following decomposition level are determined on the basis of the technical architecture.

The final step is the specification of the realization. At this most detailed level of abstraction, specifications are made for the implementation of the previously specified functional requirements, taking into account the complete technical product architecture. This type of specification addresses, for example, the definition of implementation languages used to implement software architecture elements, the detailed structuring of software models or materials used to implement hardware. The conclusion of this specification phase and at the same time the conclusion of the entire product specification, in the form of a final implementation specification, represents the transition to software implementation, hardware development and production and thus the development phase of the entire product.

The artifacts resulting from the individual steps of the procedure can be represented in textual as well as model-based form. Especially in case of a textual specification, separate artifacts can be created for each step of the procedure or a continuous extension of the artifacts can be aimed at. In the first case, the complexity of the individual artifacts is minimized, but there is a maximum number of different artifacts that can result in a complex artifact landscape. The consideration of the traceability of design decisions takes on importance as the number of artifacts increases and must be guaranteed. A continuous extension can lead to an increasing complexity and a decreasing comprehensibility and maintainability of single artifacts. Consequently, depending on the extent of the specification artifacts, a project- and system-specific decision must be made as to how many and which artifacts are to be created and to what extent an extension of artifacts is to take place over successive steps.

### 3.4 Functional system variant consideration concept

In the previous investigations, the aspect of functional system variants was not considered. However, it is very often necessary to consider system variants when specifying products and especially product lines. In the automotive industry, for example, there are different powertrain concepts (Sol) in vehicle development that are to be offered as part of a product line. The powertrain components (SE) are often used modified within different powertrain variants, but can also have functional variants. For this reason, all focused variants must be clearly defined and specified in the specification phase. The specification of the SoS remains unchanged under the assumption that only the Sol itself is variant-afflicted and the interacting systems as part of the SoS do not have any functional variants. In this case, the procedure for specification of the SoS based on customer expectations can be used in an analogous manner to the non-variant specific procedure in Chapter III.B.

On the second decomposition level, any number of variants of the Sol requires a separate consideration of each individual system variant (cf. Figure 6). For each system variant, a variant-specific, logical architecture is derived on the base of the logical architecture specification of the SoS as well as customer-specific requirements to the respective system variant, using the feature-based procedure. For the
mentioned example, a separate analysis and definition of a logical architecture must be performed for each variant of the powertrain, for example front-wheel drive, rear-wheel drive or all-wheel drive. In order to obtain the complete logical architecture of the SoI, which covers all system variants to be considered, the data of the variant-specific architectures must be combined to form a variant-independent architecture (LAS(SoI)). This architecture represents all logical architectural elements that are used both for all variants and only for a particular variant or subsets of variants.

At the following decomposition level, this results in subsets of system elements from the LAS(SoI) that can be assigned to at least one or more system variants. The more variants of the previous decomposition level can be assigned to a subset of the subsequent decomposition level, the higher is the prioritization of a maximum number of elements within this subset. This is due to the fact that the highest possible number of system elements that can be assigned to more than one subset enables system elements to be reused across variants. The specification of these system elements, regardless of whether they can be assigned to one or more system variants, is performed in parallel and analogously to the procedure for a variant-less system element (cf. Figure 5). In the case of an additional variant-related system element, a variant-specific logical architecture is defined in a first step analogous to the specification of the variant-specific SoI, and in a second step the variant-independent LAS is defined by corresponding aggregation.

The feature-based specification of the logical architecture on a decomposition level leads not only to a definition of logical architectural elements for which implementation decisions have to be made in the context of the technical architecture, but also to a definition of optional, dependent, or alternative variation points. The final set of all variation points as part of the LAS contains the technical variation points of a product as well as functional variation points through the feature-based specification of all system variants. The definition of the technical architecture includes a complete configuration of the system and a final decision for each individual variation point. For this reason, each product to be realized must have a unique specification of the technical architecture.

3.5 Application example—standardized simulation architectures of electrical vehicles

In the following section results from the research project HiFi-ELEMENTS are presented, an EU-funded project involving 19 partners from 10 different European countries. Within the project, the CUBE methodology was transferred to the development of simulation model product lines of electric vehicles.

In automotive development, simulation models are of increasing importance, especially in the early phases of validation. In recent years, a trend towards increasingly complex simulation models has been observed, as all relevant physical effects of the entire vehicle need to be simulated. The large number of possible variants of e-vehicles as well as their possible drive trains represents a second major challenge. The use
of standardized reference architectures for product lines of simulation models is one possibility in order to meet these challenges. Furthermore, the development of simulation models is characterized by interdisciplinary collaboration of experts from different domains. CUBE as a feature- and system-based methodology, was selected to address the described challenges and was used for the interdisciplinary definition of a reference architecture for X-in-the-Loop (XIL) simulation models of electric vehicles. The Electric Modeling Architecture (EleMA) is one central result of the HiFi-ELEMENTS project.

The following application example is intended to illustrate the sequence of steps in the definition of EleMA. Due to the fact that the description of the feature-based procedure on one decomposition level is sufficient to generate a basic understanding, only the decomposition level Sol is explained in detail. The SoS and SE levels are not considered in detail, but only relevant, preceding results or following steps are explained.

The model-based specification of the presented example was implemented in the form of SysML diagrams and was realized with the modeling tool Enterprise Architect. The specification of the textual requirements was realized as part of the SysML models, but a combination with a requirements management tool is also conceivable. For the sake of simplicity of the SysML diagrams, the textual requirements are not shown. For one example refer to appendix A VI.

The initial basis for the specification of the Sol is a successfully completed specification of the SoS. This includes a complete definition of the system boundaries and requirements for the Sol as well as of all systems interacting with the Sol. In the example considered, the Sol is the simulation model that represents the controlled system (plant) in a closed control loop with the system under test (SuT). It simulates all relevant characteristics of the electric vehicle, provides these as input to the SuT and in turn processes the output data of the SuT. The SuT can be any hardware component or control unit in the drive train of the electric vehicle. Based on a structured and feature-based requirements elicitation on SoS level, a set of vehicle topologies that must be considered is defined. This includes five powertrain topologies of BEVs (cf. Figure 7). These powertrain topologies cover front-, rear-, and all-wheel drive vehicles and distinguish between central and near-wheel drive engines.

The first step of the procedure for Sol specification is done in the form of a black box view of the system. It enables a solution-neutral derivation of the requirements demanded of the system by all relevant system actors. Since this is a multi-variant Sol, this consideration is done separately for each variant. For variant 3, the black box view is represented by a use case diagram (cf. Figure 8, left). The relevant actors of the simulation model are the system, software and hardware developers, the test bench operator as the primary user of the model and the test manager who coordinates the verification and validation of the SuT. All needs of the actors are documented in the form of use cases and associated requirements. For example, the test operator needs a high usability and high exchangeability of SuT and simulation models to minimize the work effort.

According to the FDD procedure, in a second step a feature list is created. Each previously determined requirement of an actor must be assigned to at least one feature. The list itself can be created both textually and model-based. A model-based specification can either be an extension of the use case diagram created in the first step (cf. Figure 8, left) or it can be created as a new view that shows the relationship between use cases and features (cf. appendix A VI). The definition of the feature itself is strongly product dependent and can be done according to different principles. The feature “Simulation of physical electromagnetic system behavior” will be considered in the following. In this example the definition of the feature was mainly made considering different simulation domains. A feature is therefore assigned to as few domains as possible. This makes it possible to reduce the coordination efforts between developers of different domains.

In a third step, the operating principle of the Sol is described from the perspective of each individual feature. To illustrate this white box view of the features, behavior diagrams of the SysML are well suited, such as activity diagrams (cf. Figure 8, right). The visualization of the operating principles of each individual feature allows a reduction of complexity compared to the behavior representation of the whole system. Furthermore, the freely adjustable definition of the feature allows to limit the behavior to be described to relevant aspects. For example, the feature definition according to simulation domains enables developers of these domains to focus on the aspects relevant to them. The considered feature describes the electromagnetic behavior of the simulation model from the source (e.g., battery) to the sink (e.g., e-motor). For reasons of simplicity, it does not show the recuperation path and the information received by or sent to other features. The last activity describes the generation of a magnetic power which can be used to generate kinetic energy. The description of the behavior is
solution-neutral, without specifying which activity is realized by which element of the system.

The fourth step is to define a logical architecture for the previously specified operating principle of each feature. The activities are assigned to individual system elements, but without defining their kind of realization (cf. Figure 8, right). Based on the object flows defined in the activity diagram, the interfaces of the system elements necessary for the realization of the feature are defined. Structure diagrams, such as internal block diagrams (IBD) (cf. Figure 9, top) are particularly suitable for representing the static aspects of the logical architecture. In this example the logical architecture consists of eight components. By combination of related controllers and controlled systems the non-functional system requirements of high modularity and low coupling were considered. The previously defined power flows are replaced here by detailed information flows.

In the fifth step, the feature-specific architectures are combined to form a complete logical architecture of this system variant. For the first time, this architecture represents the complete set of system elements and their logical interactions of this Sol variant (cf. appendix A V). By combining the feature-specific logical interfaces, the complete set of interfaces (incl. non-functional aspects of communication) for each system element of the variant-specific logical architecture can be determined (cf. Figure 9, bottom, left).

Performing the sixth step, the variant-specific architectures are combined into a multi-variant representation. The final IBD represents a variant-independent specification of each system element and its interfaces as well as the possible variation points or realization options for the implementation of a concrete system simulation (cf. Figure 9, bottom, right). Depending on the vehicle topology to be simulated, one of the represented signals is linked to the variant-independent component port when implementing a specific system simulation.

The EleMA reference architecture derived according to the presented example provides all users with a reference model including the contained components, a formalized description of the component requirements and a definition of the variant-independent component interfaces for each individual component. This serves as a starting point for the following SE decomposition level. The described procedure can be repeated iteratively on several levels of decomposition. Subsequently, a technical architecture is specified, describing architectural realization decisions for a particular product to be realized. This can be, for example, the specification of non-functional properties of the software interfaces, which must not contradict the previously defined logical architecture. In case of inconsistencies between the logical and technical architecture, the technical architecture should be adapted according to the top-down manner. If this is not possible because of failures in the logical architecture specification, these failures have to be eliminated by an iterative procedure. The final specification of the realization includes all implementation decisions for the simulation models, such as the software tools to be used. For the implementation of the simulation models in HiFi-ELEMENTS the software tools Matlab/Simulink, KULI and CarMaker were mainly used.
3.6 Definition of industrial specification processes

The presented procedure model defines a logical sequence of steps and their dependencies for the feature-based specification of product lines. For a complete process definition, in addition to the definition of responsibilities and roles, the definition of time constraints, such as milestones or the duration of activities within the specification phases, is required. In particular, these time aspects in the execution of projects are strongly dependent on the corporate culture of the executing company.

In the automotive industry, development of conventional, linear or iterative process models has been state-of-the-art in recent decades and is still widespread today. At the same time, a trend towards the development of more agile and flexible development has been discernible in large companies in various sectors for several years. In order to give both companies with conventional and agile development processes the opportunity to use the proposed methodology, it was developed with the purpose of defining a procedure model which can be used as a base for definition of conventional and more flexible, agile specification processes (addressing Objective VI).

The procedure describes steps for the feature-based development of a logical architectural definition of a decomposition element that is repeatedly applied to hierarchically ordered decomposition levels. The steps of this structured procedure can be performed completely sequentially, starting with the specification of the SoS. This results in a very thorough specification of the system during the first run through of all decomposition steps, but is at the same time very time-consuming and requires few iterations due to the high duration required to complete a cycle (cf. Figure 10). This procedure is strongly comparable with the conventional waterfall model and shows major weaknesses in the reaction to changes, such as additional or changing customer requirements and the associated specification of necessary product versions.

Alternatively, the FDD allows the specification of individual features to be parallelized. A parallelized specification of individual system
elements on different decomposition layers is also possible. All in all, this enables a faster and partially parallel processing of the specification phases. A continuous addition of functional features on the different decomposition levels while iteratively passing through the steps can allow a further increase of the iteration frequency while at the same time decreasing the specification scope. A combination of the methodology with agile methods, such as Scrum\textsuperscript{52} or SAFE,\textsuperscript{53} is therefore possible, in which functional features or decomposition elements are specified in parallel in short cycles and used to continuously expand the system specification extent. This results in improved and more flexible consideration of system versions based on changed customer requirements.

It is important to avoid increasing the specification frequency excessively, as this can lead to an excessive reduction in the number of specifications per cycle and thus to a reduction in productivity. An excessive number of parallel processed specification elements at a high frequency can result in an overload and a resulting loss of quality.

In the automotive industry, the organizational structure frequently is oriented towards the physical system architecture, and often reaches its limits when FDD is introduced. New feature-based artifacts in the development process require new responsibilities and, in particular, increased interdisciplinary cooperation between development teams. To address these challenges, it is recommended that new roles are defined in the development project. Central responsibility for the complete specification, implementation, verification and validation of a feature belongs to the feature owner, who is responsible for monitoring the workflows and quality of the work products for the feature assigned to him.\textsuperscript{38} This includes monitoring of costs and time planning in all development phases. Depending on the current focus of development or the current level of abstraction in the specification, the feature owner involves different stakeholders relevant in this context. Stakeholders are defined specifically for each feature and their number varies depending on the complexity and characteristics of the feature. Some stakeholders only need to be involved at a certain level of abstraction or only for individual work products, others at several levels. Stakeholders can have different perspectives, such as on specific development goals (e.g., safety or security), system-wide properties (e.g., architecture), or on interfaces and interactions with other systems. The interdisciplinary execution of the respective development step ends with a joint release of the work product.

All in all, FDD in large industrial projects allows an additional reduction of complexity by facilitating smaller, independent development teams.\textsuperscript{38} Secondly, interdisciplinary cooperation improves both the work product quality by involving all relevant stakeholders at an early stage and the self-commitment of the employees involved by means of joint processing.\textsuperscript{38} A third advantage is the freedom of feature definition. A definition according to development domains, as in the example in section III.E, is only one possibility of definition. This allows to actively influence the alignment efforts. Another opportunity is, for example, the definition of features that can be purchased by the end customer. The latter allows a better prioritization of development steps for companies and especially a stepwise development of products by constantly adding new system features. The decision on which principles the feature definition should be based is strongly dependent on the product and development goals and must be made on a company and project-specific basis. The advantages of FDD for industrial companies are contrasted by the challenges of a development and organizational structure that has to be changed and the definition of new roles in the development process. Furthermore, the introduction of FDD requires a change in the mindset of developers, which usually represents the greatest challenge. In accordance with the agile manifesto, the development focus shifts to interdisciplinary cooperation and the continuous addition of new system features through the introduction of FDD. Following a strict plan according to predefined processes and working in organizational structures based on the physical system architecture are replaced by new roles and structures. Furthermore, a solution-neutral development is applied, which contrasts with the classical component-based development.

4 | CRITICAL REFLECTION

The presented theoretical considerations, which led to the development of the methodology, represent an idealized, system, and feature-based top-down specification of product lines. It exclusively describes the system design procedure as part of the concept and development phase in the system life cycle. A consideration of the subsequent development phases, such as production, utilization, and retirement, in the form of a definition of steps for the implementation of these phases is not covered. At the same time, it is essential to ensure that the above-mentioned phases are taken into account in system design and can be implemented by considering the system requirements resulting from these phases. Support processes, such as supplier management or product management, are not described in detail.

Furthermore, the methodology is based on the assumption of a completely new development of the system, so that a solution-neutral consideration of the system can be carried out. However, this is often not possible or desired in industrial projects, since previous versions of the system already exist and are to be reused to the greatest possible extent. A complete new development of systems, in particular if a reuse of already existing product elements will subsequently no longer be possible, leads to significantly increasing product
5 CONCLUSION

In this contribution, a novel systems engineering methodology for the feature-driven development of product lines was systematically developed. On the base of a theoretical analysis of existing methods, procedures and current challenges in the field of SE, objectives for the derivation of an appropriate specification methodology were systematically derived. The underlying procedure model combines proven methods from the field of SE, such as stepwise partitioning of the system and structured specification by means of different system views, with the agile procedure of FDD. By combining these methods, it is possible to reduce system complexity in the specification to a greater extent than is possible with non-feature-based development. Additionally, the presented procedure covers the consideration of functional and technical variants on all defined partitioning levels. In this context, a specification of logical and reusable product line architectures is focused and an investigation of variants is conducted in order to make a decision about the best possible realization solution. As a result, the development of product lines in particular allows a reduction of development costs through optimized reusability of specification artifacts. The consideration of FDD for the specification of each individual decomposition element enables a functionally-focused view of the system. The resulting possibility of a solution-neutral consideration of the system contributes to a cost-optimized development and production of new systems, since organizational and development history related decisions are not considered during the system design. Furthermore and in contrast to existing SE procedures, the additional feature-based view of the system increases the possible granularity during development and hence creates a basis for the definition of conventional, but also flexible, agile development processes. The resulting artifacts of the individual steps of the procedure were used exemplarily for the development of a reference architecture for simulation models of electric vehicles (Electric Modeling Architecture—EleMA)

In the future a more detailed presentation of the application of the methodology for the model-based and textual specification of systems is planned. Especially, the design of new system generations based on already existing system versions has to be investigated in more detail, since an idealized top-down specification is not possible in this case. Furthermore the combination of the presented methodology with scenario-based development for automated or fully autonomous driving functions is already under investigation. In addition, the quantification of effort reductions by applying the new methodology is to be carried out after the successful completion of a large number of development projects.

ACKNOWLEDGEMENT

The authors thank all partners of the HiFi-ELEMENTS project.
FIGURE 11  Flag of European Union

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 769935.

Open access funding enabled and organized by Projekt DEAL.

DATA AVAILABILITY STATEMENT

The data that supports the findings of this study are available in the supplementary material of this article.

ORCID

Christian Granrath M.Sc.  https://orcid.org/0000-0002-9793-1973
Sebastian Silberg B.Sc.  https://orcid.org/0000-0002-2985-2798
Max-Arno Meyer M.Sc.  https://orcid.org/0000-0003-4420-0357
Prof. Dr.-Ing. Jakob Andert  https://orcid.org/0000-0002-6754-1907

REFERENCES

1. Xue R, Baron C, Esteban P, et al., “Alignment of practices for an efficient management of Systems Engineering processes during the development of systems of systems,” 2015 Third World Conference on Complex Systems (WCCS), 2015 Third World Conference on Complex Systems (WCCS), Marrakech, Morocco, 23.11.2015 - 25.11.2015, IEEE, ISBN 978-1-4673-9669-1:1-6, 2015 - 2015.
2. Conforto E, Rossi M & Rebentisch E, et al., “Survey report: Improving integration of program management and systems engineering,” 2013.
3. Kriebel S, Moyzes V, Strobl G, et al. The next generation of bmw’s electrified powertrains: Providing software features quickly by model-based system design,” 26th Aachen Colloquium Automobile and Engine Technology, Aachen, 2017.
4. Kriebel S, Richenhagen J, Granrath C, Kugler C. Systems engineering with SysML - the path to the future? MTZ Worldwide. 2018;79(5):44-47.
5. Martinez de Aragón B, Alonso-Zarate J, Layá A. How connectivity is transforming the automotive ecosystem. Internet Techn. Lett. 2018;1(1):e14.
6. Wortmann F, Flüchter K. Internet of things. Bus Inf Syst Eng. 2015;57(3):221-224.
7. Schulz C. Agile Software-Produktlinienentwicklung im Kontext heterogener Projektlandschaften. Shaker. 2019. ISBN 9783844066838.
8. Schaufele J, Zuraufka T, “Automotive Software Engineering: Principles, Processes, Methods and Tools,” 2nd ed., SAE International; 2016, ISBN 9780768079920.
9. Broy M, Krcmar H, Zimmermann J, Kirstan S. Economical impact of model-based development of embedded software systems in cars. ATZelektronik worldwide eMagazine 62. 2011;6(2):18-21.
10. Kriebel S, (ed.), “Economic High Quality Software for Automotive Systems,” 3rd Congress on Real Time, Braunschweig: 2011.
11. Walden DD, Roedler GJ, Forsberg K, Hamelin RD, et al. Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities. 4th ed. Hoboken, New Jersey: John Wiley & Sons, Inc.; 2015. ISBN 9781118999400.
12. Esteban J. Survey of model-based systems engineering (MBSE) methodologies. In: MBSE Focus Group 25.8; 2007;1-12.
13. Grönninger H, Hartmann J, Krahn H, Kriebel S, et al. View-Centric Modeling of Automotive Logical Architectures. Informatik-Report 2008-02, CFG-Fakultät, TU Braunschweig; 2008.
14. Bahill AT, “Model-Based Systems Engineering,” CRC Press; 2018. ISBN 9780203746936.
15. Bohmer AI, Hugger P, Lindemann U. “Scrum within hardware development insights of the application of scrum for the development of a passive exoskeleton,” 2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC), 2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC), Funchal, 27.06.2017 - 29.06.2017, IEEE, ISBN 978-1-5386-0774-9:790-798, 2017 - 2017.
16. Klein TP, Reinhart G. Approaches for integration of agile procedures into mechatronic engineering of manufacturing systems. In: Zaeh M, ed. Enabling Manufacturing Competitiveness and Economic Sustainability, band 319. Cham: Springer; 2014. ISBN 978-3-319-02054-9.
17. Beck K, Beedle M, van Bennekum A, et al, “Manifesto for Agile Software Development”; 2019. https://agilemanifesto.org/en/manifesto.html.
18. BKCASE Editorial Board (ed.), “Guide to the Systems Engineering Body of Knowledge (SEBoK): v 1.4,” BKCASE is managed and maintained by the Stevens Institute of Technology Systems Engineering Research Center, the International Council on Systems Engineering, and the Institute of Electrical and Electronics Engineers Computer Society, 2015.
19. International Organization for Standardization, “Systems and Software Engineering - System life cycle processes,” ISO/IEC/IEEE 15288:2015(E), ISO copyright office, Rev. May, 2015.
20. Project Management Institute, A Guide to the Project Management Body of Knowledge: (PMBoK® Guide) – Fifth Edition,” 5th ed., Project Management Institute, Inc., Pennsylvania, USA, ISBN 978-1-935589-67-9, 2013.
21. Bloomberg J, Schmelzer R, “Service Orient or Be Doomed!” Hoboken, New Jersey: John Wiley & Sons, Inc., 2006. ISBN 0471768588.
22. International Organization for Standardization, “Systems and software engineering — Vocabulary,” ISO/IEC/IEEE 24765:2017(E), ISO copyright office, Rev. Sep. 2017.
23. International Organization for Standardization, “Systems and software engineering — Requirements for designers and developers of user documentation,” ISO/IEC 26514: 2008(E), ISO copyright office, Rev. Jun. 2008.
24. International Organization for Standardization, “Quality management systems — Fundamentals and vocabulary,” ISO 9000:2015(en), ISO copyright office, Rev. Nov. 2015.
25. International Organization for Standardization, “Systems and software engineering — Software life cycle processes,” ISO/IEC/IEEE 12207:2017(E), ISO copyright office, Rev. Nov. 2017.
26. "IEEE Standard for Software Quality Assurance Processes," 730-2014, IEEE, Rev. Jun. 2014.
27. International Organization for Standardization, “Information technology — Systems and software engineering — Guide for requirements engineering tool capabilities,” ISO/IEC TR 24766:2009(E), ISO copyright office, Rev. Dec. 2019.
28. International Organization for Standardization, “Information technology — Modeling Languages — Part 2: Syntax and Semantics for IDEF1X97 (IDEFobject),” ISO/IEC/IEEE 31320-2:2012(en), 1st ed., ISO copyright office, Rev. Sep. 2012.
29. Brinkkemper S. Method engineering: engineering of information systems development methods and tools. Inform. Softw. Technol. 1996;38(4):275-280.
30. "IEEE Standard for Conceptual Modeling Language Syntax and Semantics for IDEF 1X/Sub 97/(IDEF/Sub Object)," 1320-2-1998, IEEE, Rev. 1998.
31. Kruchten P. Rational Unified Process, the: An Introduction. 3rd ed. Addison-Wesley Professional, Boston. ISBN 9780321197702, 2003.
32. International Organization for Standardization. “Systems and software engineering — Content of life-cycle information items (documentation),” ISO/IEC/IEEE 15289:2017(E), ISO copyright office, Rev. May, 2017.
33. Kossiakoff A, Sweet W, Seymour S, Biemer S. Systems Engineering: principles and Practice. Wiley Series in Systems Engineering and Management. 2nd ed. Chichester: John Wiley and Sons; Hoboken, New Jersey, 2011. ISBN 9780470405482.
34. Haberfellner R, Fricke E, Weck O, Vössner S. “Systems Engineering: Grundlagen und Anwendung,” 13th ed., Orell Füssli Verlag AG, Zürich, ISBN 978–3280040683, 2015.
35. Friedenthal S, Moore A, Steiner R. A Practical Guide to SysML: Systems Model Language. Burlington, Mass: Elsevier/Morgan Kaufmann; 2008. ISBN 978–0123743794.
36. Pohl K, Hönninger H, Achatz R, Broy M. Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology. Berlin, Heidelberg: Springer; Imprint: Springer; 2012. ISBN 978-3-642-34614-9.
37. Pohl K, Broy M, Daembkes H, Hönninger H. Advanced Model-based Engineering of Embedded Systems: Extensions of the SPES 2020 Methodology. Cham: Springer International Publishing; 2016. ISBN 978-3-319-48003-9.
38. Ewald J, Orth P, Granrath C, Andert J, “Model Based Systems Engineering for Standardized Simulation Frameworks: Case Study Development of Electrical Vehicles,” in: 41st International Vienna Motor Symposium, 2020.
39. Turner R. Toward agile systems engineering processes. Crosstalk J. Defense Softw. Eng. 2007:11-15.
40. Douglass BP. Agile Systems Engineering. Waltham, MA: Morgan Kaufmann; 2016. ISBN 9780128021200.
41. Dove R. Agile systems-engineering and agile-systems engineering. Insight Int Council Syst Eng. 2004;17(2):6-10.
42. Palmer SR, Felsing JM. A Practical Guide to Feature-Driven Development. Upper Saddle River, New Jersey: Prentice Hall PTR; 2002. ISBN 978–0130676153.
43. Kang KC, Lee J, Donohoe P. Feature-oriented product line engineering. IEEE Softw. 2002;19(4):58-65.
44. Chowdhury AF, Huda MN. “Comparison between Adaptive Software Development and Feature Driven Development,” Proceedings of 2011 International Conference on Computer Science and Network Technology, 2011 International Conference on Computer Science and Network Technology (ICCCSNT), Harbin, China, 24.12.2011 - 26.12.2011, IEEE, ISBN 978-1-4577-1587-7:363–367, 2011-2011.
45. Linden F, Schmid K, Rommes E. Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. 1st ed. Springer-Verlag; Berlin Heidelberg. 2007 s.l., ISBN 978-3-540-71437-8.
46. Clements P, Northrop L. Software Product Lines: Practices and Patterns. SEI Series in Software Engineering. Boston, London: Addison-Wesley; 2002. ISBN 9780134424088.
47. Alqudah M, Razali R. A review of scaling agile methods in large software development. Int. J. Adv. Sci. Eng. Inform. Technol. 2016;6(6):828.
48. Kruchten PB. The 4+1 view model of architecture. IEEE Softw. 1995;12(6):42-50.
49. Maier MW. Architecting principles for systems-of-systems. Syst Engin. 1998;1(4):267-284. https://doi.org/10.1020/1520-6858(1998)1.
50. Selva D, Cameron B, Crawley E. Patterns in system architecture decisions. Syst Engin. 2016;19(6):477-497.
51. Granrath C, Meyer M, Andert J, Ewald J, et al. EleMA: a reference simulation model architecture and interface standard for modeling and testing of electric vehicles. eTransportation. 2020(4).
52. Schwaber K. SCRUM development process. In: Sutherland J, Casanave C, Miller J, Patel P, eds. Business Object Design and Implementation. London: Springer; 1997. ISBN 978-3-540-76096-2:117–134.
53. Leffingwell D. SAFe 4.5 Reference Guide: Scaled Agile Framework for Lean Software and Systems Engineering. Boston: Addison-Wesley; 2018. ISBN 978–0134892863.
54. Eisner H. “Essentials of Project and Systems Engineering Management.” 3rd ed., J. Wiley, Hoboken, N.J., ISBN 978–0470129333, 2008.

AUTHOR BIOGRAPHIES

Christian Granrath, M.Sc. received the B.Sc. degree in mechanical engineering in 2014 and the M.Sc. degree in energy engineering in 2016 from RWTH Aachen University, Aachen, Germany. From 2016 to 2021 he performed his doctoral studies at the Junior Professorship for Mechatronic Systems for Combustion Engines, Institute for Combustion Engines, RWTH University. Since 2021, he is working as a technical specialist for systems engineering at FEV Europe GmbH. His research interests include the fields of model- and feature-based systems engineering, agile software engineering, software architecture development and evaluation as well as simulation model development for XIL applications in automotive domain. Mr. Granrath’s awards and honors include the Dean’s List, representing the top 5% of the best students in each year.

Christopher Kugler, M.Sc. received his master in Computer Science at the RWTH Aachen University in 2014. From 2015 to 2019 he performed his doctoral studies at the computer science chair “Informatik 11 – Embedded Software” at the RWTH Aachen University. His doctorate thesis investigates how the increase in complexity in automotive product development can be handled by means of risk-based verification & validation approaches. Since 2019 he is working as a team leader for Systems Engineering & Test Management at FEV Europe GmbH. The focus of his work lies on the definition and implementation of model-based systems engineering practices as well as verification & validation concepts for automotive customers, where a feature-centric and risk-driven approach is promoted by him and his team.
Sebastian Silberg, B.Sc. received the B.Sc. degree in Information Systems in 2016 from WWU University of Münster, Germany. He is currently pursuing a second study in Business Administration and Mechanical Engineering at RWTH Aachen University, Germany. In parallel, he is employed at FEV Europe GmbH within the business unit Intelligent Mobility & Software as member of the systems engineering department. He is involved in several projects and research with the emphasis on Systems Engineering.

Max-Arno Meyer, M.Sc. received the B.Eng. degree in vehicle and propulsion technology in 2015 and the M.Sc. degree in international automotive engineering in 2018 from FH Aachen University of Applied Sciences, Aachen, Germany. He is currently pursuing the Ph.D. degree in software engineering at the Junior Professorship for Mechatronic Systems for Combustion Engines, Institute for Combustion Engines, RWTH Aachen University. His research interests include the fields of simulation-based testing of embedded systems, ADAS and autonomous driving, agile software engineering as well as model- and feature-based systems engineering. Mr. Meyer has been awarded the Honorary Plaque of FH Aachen for his academic achievements being among the top five percent of the best students in each year.

Dr.-Ing. Philipp A. Orth received his Diploma Degree with distinction in Mechanical Engineering in 2000. In 2005, he obtained a Doctorate Degree in the same faculty of RWTH Aachen University at the Institute of Automatic Control. In his doctorate thesis, he investigated the applicability of Rapid Control Prototyping methods to Industrial Automation, bringing together control analysis and synthesis of discrete controllers with programming of programmable logic controllers (PLC) in an integrated SW tool chain. From 2006 to 2007, he worked at ABB corporate research on engineering efficiency and tool chains for process control systems and safety PLCs. Since 2007, he has worked as team lead at FEV until mid of 2014, responsible for model-based software development of embedded control systems for automotive and/or powertrain series applications, before changing to the position of senior project manager for SW and systems engineering in the context of heavy-duty and passenger-car applications. Since 2019 he leads the department Systems Engineering within the Business Unit Intelligent Mobility & Software at FEV Europe GmbH, focusing on improvements in SE methodology and establishing SE within FEV as well as at – mainly automotive - OEM and Tier-1 customers. He is author and co-author of more than 30 articles, conference presentations and patents on the mentioned topics.

Dr.-Ing. Johannes Richenhagen received the Diploma Degree in Industry Engineering & Management in 2009. In 2014, he obtained a Doctorate Degree in Mechanical Engineering, both from the RWTH Aachen University. In his doctorate thesis, he investigated the applicability of Agile Software Development methods to pow- ertrain control software development to close the gap between engineering and computer sciences for future mobility applications. From 2013 to 2018, he was working as a technical specialist, team leader and department manager at FEV Europe GmbH. He focused on the design and test of embedded software for automotive systems. Since 2019, he leads the Business unit Intelligent Mobility & Software at FEV Europe GmbH. His responsibilities include connected mobility, automated and assisted driving, systems engineering, functional safety and cyber security, vehicle electronics, software development and Data Science. He is author and co-author of more than 50 articles and conference presentations around the mentioned topics. Since 2015, he is as guest lecturer at RWTH Aachen University for ”Software Development for Combustion Engines.”

Prof. Dr.-Ing. Jakob Andert received his Dipl.-Ing. and Dr.-Ing. degrees at the RWTH Aachen University in 2007 and 2012, respectively. From 2012 to 2014, he was a project manager at FEV GmbH in the field of hybrid and electrical drive trains. Since 2014, he is Junior Professor for mechatronic systems in combustion engines at the Institute for Combustion Engines (VKA) at RWTH Aachen University. He received his Dr.-Ing. degree for a dissertation about a real time cycle-to-cycle control of homogeneous charge compression ignition engines. The Junior Professorship addresses mechatronic system in combustion engines, such as electrical cam timing actuators as well as real time systems, FPGA indication systems and Vehicle-to-X communication systems.

How to cite this article: Granrath C, Kugler C, Silberg S, et al. Feature-driven systems engineering procedure for standardized product-line development. Systems Engineering. 2021;24:456–479. https://doi.org/10.1002/sys.21596
FIGURE A I. Overview of SMArDT layers

FIGURE A II. Structure of SPES modeling framework
FIGURE A III. BWI Hall Concept – Modul 1 and 2 (translated from German)\textsuperscript{34}

FIGURE A IV. BWI Hall Concept – Modul 3 and 4 (translated from German)\textsuperscript{34}
FIGURE A V. Logical Architecture of BEV plant simulation (Sol) variant 3
**Figure A VI.** Excerpt of model-based feature list incl. textual requirements