Comput Sci Res Dev (2014) 29:21–43
DOI 10.1007/s00450-010-0136-y

REGULAR PAPER

Guiding requirements engineering for software-intensive embedded systems in the automotive industry

The REMsES approach

Peter Braun · Manfred Broy · Frank Houdek · Matthias Kirchmayr · Mark Müller · Birgit Penzenstadler · Klaus Pohl · Thorsten Weyer

Received: 19 October 2009 / Accepted: 6 September 2010 / Published online: 20 October 2010

Abstract Over the past decade, a dramatic increase of functionality, quantity, size, and complexity of software-intensive embedded systems in the automotive industry can be observed. In particular, the growing complexity drives current requirements engineering practices to the limits. In close cooperation between partners from industry and academia, the recently completed REMsES (Requirements Engineering and Management for software-intensive Embedded Systems) project has developed a guideline to support requirements engineering processes in the automotive industry. The guideline enables the requirements engineers to cope with the challenges that arise due to quantity, size and complexity of software-intensive systems. This article presents the major results of the project, namely, the fundamental principles of the approach, the guideline itself, the tool support, and the major findings obtained during the evaluation of the approach.

Keywords Software engineering · Requirements · Specifications · Methodologies · Tools

Partly funded by the German Federal Ministry of Education and Research (BMBF) project REMsES under grant No. 01 IS F06 D.

P. Braun
Validas AG, Arnulfstr. 27, 80335 München, Germany
e-mail: peter.braun@validas.de

M. Broy · B. Penzenstadler (✉)
Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany
e-mail: penzenstadler@tum.de

M. Broy
e-mail: broy@tum.de

F. Houdek · M. Kirchmayr
Daimler AG, Wilhelm-Runge-Str. 11, 89081 Ulm, Germany

F. Houdek
e-mail: frank.houdek@daimler.com

M. Kirchmayr
e-mail: matthias.kirchmayr@daimler.com

M. Müller
Robert Bosch GmbH, Postfach 300240, 70442 Stuttgart, Germany
e-mail: mark.mueller2@de.bosch.com

K. Pohl · T. Weyer
Universität Duisburg-Essen, Gerlingstr. 16, 45127 Essen, Germany

K. Pohl
e-mail: klaus.pohl@sse.uni-due.de

T. Weyer
e-mail: thorsten.weyer@sse.uni-due.de

1 Introduction

As one of the core disciplines of any engineering process for technical systems, requirements engineering aims at systematically specifying the requirements for the systems to be developed. This ought to be done in a way that the subsequent engineering activities (e.g. architecture design, detailed design, implementation, and test) are supported in the best possible way. In order to achieve the aforementioned goal and in order to guarantee that the realized system satisfies the intentions of the system's stakeholders and considers the constraints of the environment, the requirements specification has to exhibit specific quality criteria, e.g. completeness, correctness (cf. [27]).

1.1 Software-intensive embedded systems

The notion "software-intensive embedded system" is defined by combining the meanings of "embedded system" and “software-intensive system”. An embedded system is a technical system[1] that operates in a physical and technical environment. Systems of that type measure or control their environment using variables that reference physical or technical properties of the environment (cf. [43]). The IEEE Std 1362-1998 defines the notion “software-intensive system” as a system “*for which software is a major technical challenge and is perhaps the major factor that affects system schedule, cost, and risk*” [26]. Additionally, [26] states that typical software-intensive systems consist of software as well as of hardware.

Software-intensive embedded systems are widely spread in our daily life. They can be found in many real world domains such as automation technology, medical technology, telecommunications, consumer electronics, avionics, and in the automotive domain. Software-intensive embedded systems in modern vehicles, for example, provide for comfort of the occupants or ensure their safety. Typical systems of this kind are electronic stability systems, vehicle light systems, engine management systems, pre-crash detection systems, adaptive cruise control systems or traffic lane assistance systems.

1.2 Observations in the automotive industry

In the automotive industry, engineering processes for software-intensive embedded systems go along with three fundamental challenges we observed in many industrial projects.[2]

1.2.1 Increasing size and complexity

Especially over the past 10 years, a significant increase of the size and complexity of software-intensive embedded systems can be observed. The number of software-based functions in vehicles increased permanently over the past decade. As reported in [46], a typical upper class vehicle in 2007 contained about 270 implemented functions to interact with its driver. Upper class vehicles of the most recent generation contain more than 500 of such functions.

Not only did the number of software-based functions grow but also the size of the software of vehicles grew significantly over the past 10 years. The amount of binary code in an upper class vehicle in 2007 was about 65 megabyte. The current generation of such vehicles requires more than one gigabyte of binary code (cf. [46]).

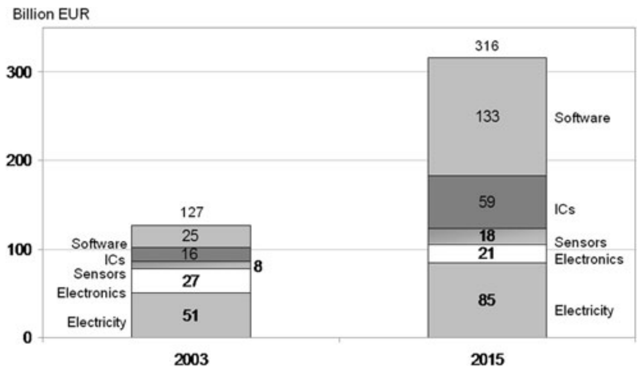


Fig. 1 Market volume of software in the automotive domain

The increasing number and size of software-based functions comes along with a growth of complexity.[3] The perceivable functions of a vehicle are increasingly realized by integrating fine-grained software-intensive embedded subsystems. Therefore the complexity of software-intensive embedded systems in vehicles increases in two ways (cf. [21]):

– *Intra-system complexity:* Since the customer requirements regarding well-known functionalities increasingly require more individual software-intensive systems (e.g. the breaking system, engine management, driving assistance system), manifold and more sophisticated features are required. That leads to a higher degree of complexity of these individual systems.
– *Inter-system complexity:* More and more perceivable functions in modern vehicles are realized by integrating systems into higher compound structures that collaboratively provide extensive services (“systems of systems”). Therefore, the number and variety of inter-system relationships increases significantly. Consequently, in addition to the internal complexity of a system, the complexity of the relationships to other systems becomes more and more important to manage.

1.2.2 Increasing economical relevance

A joint study [10] of the Mercer Management Group, the Fraunhofer-Gesellschaft, and the Robert Bosch GmbH predicted that the market volume of software in the automotive domain would have a high growth rate until 2015. Figure 1 compares the market volume of software in the automotive domain in 2003 with the predicted market volume for 2015.

As shown in Fig. 1, the study predicts a growth of the market volume of software from 25 billion EUR in 2003 to 133 billion EUR in 2015. This is equivalent to an increase

[1]A technical system can be defined as a system that is built by means of technical resources which collaborate in order to achieve an overall system goal.

[2]However, these observations are also valid for the majority of other domains.

[3]The term *complexity* of a system refers to the number of different building blocks and the number of non-trivial relationships between these blocks.

of the market volume of software in the automotive domain of nearly 530%. These market trends rely on the estimations that, beginning in 2010, more than 40% of value creation in automotive industry is realized by software and that the majority of innovations in the automotive industry will be realized by software (cf. [22]). In [19], experts from Daimler AG prognosticate that almost 80% of all future automotive innovations will be on the basis of software. Consequently, the negative effects of software project failures in the automotive industry or system failures of software-intensive embedded systems in the field seriously impact the economic success of the responsible companies.

1.2.3 Inappropriate requirements engineering

Our experiences show that requirements engineering processes in the automotive industry are not well-defined in most cases. As a consequence, engineering staff that is involved in the elicitation, specification, and quality assurance of requirements mostly proceeds in an ad-hoc manner. Requirements for a new system are mostly documented in the form of natural language statements and are usually structured by predefined requirements templates (cf. [54]).

In view of the increasing size and complexity of software-intensive systems and the growing economic importance of software-based functions in vehicles, the *current practice* with respect to requirements engineering in the automotive industry is often inadequate for coping with these upcoming challenges (cf. [2, 3, 54]).

1.3 The REMsES project

The project was partly funded by the German Federal Ministry of Education and Research (BMBF) in the context of the research initiative "Software Engineering 2006". The acronym "REMsES" stands for Requirements Engineering and Management for software-intensive Embedded Systems.

1.3.1 Vision of the project

The vision of the project was to *develop a field-proven process guide for supporting requirements engineering processes in the automotive industry*. Whenever possible, this should be done by using techniques which are already established in industrial practice rather than compete against the current state of practice.

Motivated by the three fundamental observations made in requirements engineering processes within the automotive industry, the REMsES Guide aims at enabling engineers to cope with: (1) the growing quantity and size of software-intensive embedded systems in vehicles, (2) the increasing inter-system complexity, and (3) the increasing intra-system complexity within large compound structures.

1.3.2 Project structure

The project started in August 2006 and was completed in August 2009. The project duration of 36 month was divided into two coarse-grained project phases:

- *Phase I: Analysis (month 1–3):* Requirements and associated application scenarios for the REMsES Guide were elicited from the industrial partners based on an adaptation of the strategy proposed in [47]. Afterwards, the requirements and application scenarios were analyzed, consolidated and documented within a *requirements document*. The application scenarios were the main inputs for the subsequent evaluation activities.
- *Phase II: Development and Evaluation (month 4–36):* Based on the requirements document, in this phase the guide was developed. Complementary to the development of the guide, tools supporting the application of the guide in industrial environments were designed. Accompanying the development of the guide in three engineering-evaluation cycles, several evaluation activities in academic and industrial environments were conducted. The results of the industrial evaluation were analyzed in order to define the required changes and extensions that had to be integrated in the guide within the next engineering iteration.

1.3.3 Members of the project consortium

The guide was developed in the project in close collaboration between academic partners, industrial partners from the automotive industry, and industrial partners whose main focus lies on consulting the automotive industry with regard to model-based specification and formal techniques for quality assurance. The project consortium consists of two academic and three industrial members.

Academic members

- Technische Universität München
- Universität Duisburg-Essen

Industrial members

- Daimler AG (Vehicle Manufacturer)
- Robert Bosch GmbH (First Tier Supplier)
- Validas AG (Consulting Company)

1.4 Challenges from industrial partners

At the beginning of the project, the industrial partners took lead in gathering the major industrial application scenarios concerning the support of requirements engineering processes in the automotive industry. The analysis of the gathered application scenarios revealed four major challenges.

The guide and the complementary tool support should:

- consider input information as well as other information that must be documented;
- provide fine-grained, stepwise instructions for documentation activities;
- provide templates and examples and recommend documentation techniques;
- be tailorable to project-specific or company-specific needs.

1.5 Outline

The remainder of this article is structured as follows: Sect. 2 introduces the four core principles of the approach. In Sect. 3, the major elements of the guide are exemplified. Afterwards, Sect. 4 elaborates on the tool support for efficiently using the guide in practice. The evaluation strategy and important evaluation results are presented in Sect. 5. A discussion of the related work is given in Sect. 6. The article ends with a conclusion and an outlook on future work in Sect. 7.

2 The four principles of the approach

The approach is based on fundamental principles.[4] The principles provide a foundation for any activity performed during the requirements engineering processes. The principles are:

- *Consider system decomposition explicitly*
- *Distinguish between problem and solution*
- *Keep the documentation model-based*
- *Focus on artefacts*

These four principles aim at establishing specific ways of thinking in order to meet the increasing number, size, and complexity of software-intensive systems in the automotive industry (see Sect. 1). The approach realizes each of these principles by introducing appropriate conceptualizations and techniques. Each of the four principles is explained below.

2.1 Consider system decomposition explicitly

The objective of this principle is to consider the decomposition of the overall system into fine-grained subsystems. Along the decomposition of the system, its engineering process can be divided into a number of individual fine-grained engineering processes, complemented by activities that support the integration of the results.

[4]The term *principle* is used in the sense of [17] as a convention that is subject to any activity.

2.1.1 Abstraction layers in general

Requirements engineers in the domain of software-intensive embedded systems tend to specify the requirements at a detailed, technical level. Yet, the documentation of high-level requirements is insufficient (cf. [54]). The higher levels of abstraction are essential for justifying detailed requirements, understanding a requirements document, and managing the high complexity of embedded systems. Requirements at different levels of detail and abstraction, ranging from goals to detailed technical requirements (e.g. concerning the system hardware), need to be included in the requirements document. High-level requirements provide a justification for detailed requirements and support the understandability of the requirements document. Low-level (i.e. detailed) requirements are needed to provide enough information for implementing the system correctly.

The diversity of requirements at different levels of detail demands a systematic way of dealing with each requirement adequately and dealing with it according to its level of detail and its granularity. The granularity of a requirement influences, for instance, its importance in a certain stage of system development. For example, at an early stage of contract negotiation, coarse-grained goals might receive more attention than fine-grained and detailed technical requirements. Later on, when the feasibility of the planned system is evaluated, detailed technical requirements or even a preliminary high-level system design are needed. A well-known technique for dealing with varying levels of detail and granularity of requirements is to establish multiple layers of abstraction and assign each requirement to the appropriate abstraction layer (cf. e.g. [18]).

2.1.2 The REMsES abstraction layers

In literature as well as in industrial practice, different hierarchies of abstraction layers for technical systems are defined and used. Existing approaches like [15, 38] vary, for example, in the way how the abstraction layers are generated and in the number of resulting layers. In the project, we have adopted a hierarchy of three abstraction layers (see Fig. 2): system layer, function groups layer, and hardware/software layer.

The *separation between system layer and function group layer* is based upon the academic and practical experiences that complexity and size of technical systems can be mastered by decomposing the system in a set of coherent logical components. Consequently, the overall engineering problem is divided into a number of small engineering problems together with the problem of integrating the resulting logical components of the system.

The *separation between function group layer and hardware/software layer* is based upon academic and practical

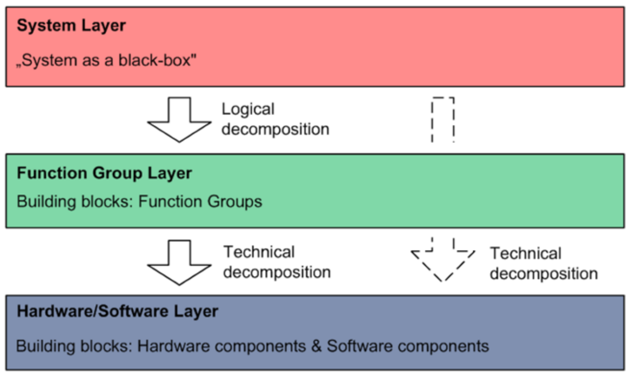


Fig. 2 Abstraction layers and system decomposition

experiences that the logical decomposition of a software-intensive embedded systems should be independent from the decisions concerning which aspects of the systems are realized by software and which aspects are realized by hardware. The characteristics of each of the three abstraction layers are described below.

– *System layer:* At the system layer, the stakeholders take an "outside" view (i.e. black box view) of the system. Models on this layer focus on how the system is used by its users as well as by other systems. This layer comprises, for instance, usage goals, usage scenarios, and the functions or services that the system offers to its users via predefined interfaces. These services represent the functionality that is directly visible to the users. As a consequence, this layer does not regard system-internal functions.
– *Function group layer:* The function group layer represents a whitebox view of the system. At this layer, the system is regarded as a network of interacting, logical units obtained by a logical functional decomposition of the system. We refer to these units as "function groups". The information captured at the system layer can be refined at the function group layer and assigned to individual function groups. Function groups have defined interfaces and can interact with each other as well as with the system environment. Each function group exhibits a defined behavior at its interfaces. Function groups are identified, for instance, by logical decomposing and clustering the system functions that are defined at the system layer.
– *Hardware/software layer:* At this layer, a physical partitioning of the system functionality into hardware and software is defined. For this purpose, the system is technically decomposed into individual hardware and software components. This decomposition can be regarded as a preliminary or draft technical system architecture. Software components may include code of more than one function group. Hardware components are devices such as electronic control units (ECUs), sensors and actuators. A hardware component can be used by more than one function group. It should be noted that detailed design models are not within the scope of the hardware/software layer. The main purpose of such models in the REMsES approach is to support the refining and specification of requirements (cf. [41]).

How the abstraction layers address the challenges Abstraction layers help to control the high system complexity that can be found in software-intensive embedded systems, for example, when decomposing a software-intensive embedded system of a vehicle (e.g. an adaptive cruise control) into logical subsystems (e.g. control unit, signal processing, sensors, actuators). In addition, abstraction layers allow for a clear assignment of requirements, for example the requirement "*In case of . . . the system shall emit . . .*" clearly belongs to another abstraction layer than the requirement "*The software function F123 and F124 may not be executed on the same processor*". The abstraction layers support separation of concerns such as differentiating between external and internal system behavior, or the distinction between functional decomposition and the partitioning into hardware and software. Finally, the abstraction layers provide assistance for verification, for example, for a security certification. Such a certification may require that each component is justified by a system requirement.

2.2 Distinguish between problem and solution

The objective of this principle is to continuously distinguish between the information concerning the underlying problem and information concerning the corresponding solution, with regard to the systems to be developed. This principle represents one of the core principles of requirements engineering (cf. [57]). The problem description elaborates on the underlying engineering problem with regard to characteristics and conditions of the environment (in which the system shall operate in the future) together with the goals and needs of the stakeholders, and aspects that constrain the characteristics of the system. The documentation of the solution contains information about how to solve the problem stated in the problem description.

2.2.1 The distinction "environment", "system", and "interface" in general

The differentiation between the problem and the solution leads to three distinct system views. According to Jackson's "World and the Machine" model [29], there are three major views onto a system: Firstly, there are requests for changes that the system shall enforce in the *world* (i.e. in its environment). Secondly, there are characteristics the system shall exhibit at its *interfaces* to the world. Thirdly, there are specifications for the "inside" of the machine or system, documenting details about how the system should be realized.

The environment as well as the conditions and capabilities the system should provide at its interface define the problem that has to be solved by the system. Although their is a notational similarity between the system view "system" and the abstraction layer "system layer", both classifications are orthogonal to each other, e.g. the system view "system" can be applied to each of the three abstraction layers described in Sect. 2.1.2.

2.2.2 *The system views*

Based on the distinction between environment, interface, and system, the approach distinguishes between three main content categories: "context", "requirements", and "design". The "world" is modelled in the *context*, the "interface" is captured in the *requirements*, and the "inside" is represented by the *design*. The categories "context" and "design" contain important information for the requirements engineering process and therefore have a considerable influence on the requirements (cf. [41, 52]).

– *Content category "context":* The context of the system is the part of the real "world" that influences the requirements for the system and therefore the system itself. Context elements are, for example, laws, business goals, general constraints, technical or physical environmental conditions, and information about adjacent systems. Many requirements for the system and its components, respectively, originate directly from the demands and constraints imposed by the context (cf. [55]). The context of a system has an inner boundary and an outer boundary. The inner boundary (system boundary) separates the system from its environment and therefore defines the subject of engineering. The outer boundary (context boundary) separates the part of the environment that influences the requirements of the system from the irrelevant part of the world that does not.
– *Content category "requirements":* The content category "requirements" defines conditions or capabilities the system should exhibit at its interface. Information within this content category can be documented using both natural language and conceptual models such as goal models (cf. e.g. [35]), scenario models (cf. e.g. [8]), or models of function, data, or behavior (cf. e.g. [12]). In the industrial context of the project partners, goals, scenarios, and functions were identified as the most important concepts of a software-intensive embedded system within the content category "requirements".
– *Content category "design":* As already stated above, a certain amount of design information in the requirements document of an software-intensive embedded system is inevitable, because the consideration of major design decisions is required to specify detailed requirements (cf. [41]). Detailed requirements such as component requirements must be specified, for instance, to facilitate the integration of systems developed by different suppliers.

2.2.3 *Abstraction layers and content categories*

The two principles described so far are orthogonal to each other. In other words, the three content categories are defined orthogonally to the abstraction layers. The abstraction layers form the vertical dimension of the structure and the content categories relate to the horizontal dimension of the model structure.

How the system views address the challenges We have found that in industrial practice, requirements often contain implicit design choices, which makes the requirements unnecessarily complex and unnecessarily limit the solution space. In such cases stakeholders may not be aware which of the so called "requirements" hide design choices and which ones are actually part of the content category "requirements". Furthermore, developers need to understand where certain information has to be documented. Distinguishing between the three content categories of information helps to reduce complexity by separating the indicative problem description (context), the optative problem description (requirements) and the description of the solution (design).

2.3 Keep the documentation model-based

The objective of this principle is to establish a continuous model-based documentation of information that is created during the requirements engineering process. The notion "model-based" is used in the approach in two ways (cf. [13]):

– The content structure of the relevant information is defined by a meta-model that specifies the pieces of information and the structural dependencies between them.
– The relevant information is documented using visual conceptual modeling languages (e.g. class diagrams, message sequence charts, and statecharts).

Since conceptual modeling languages are typically defined using meta-models that can be regarded as specific information models, the two interpretations of the notion "model-based" are not disjoint. The interpretations differ with respect to the fact that the template-based documentation refers to "model-based" in the sense that the template structure is visible in the form of an information model. Therefore, the template-based documentation is regarded as a kind of "model-based documentation".

2.3.1 *Continuous modelling in general*

The level of automated support that can be provided for natural language requirements is limited. Thus, requirements engineering tasks such as requirements analysis, validation, or negotiation become increasingly difficult when the size and complexity of the requirements specification grows. As pointed out in [56] model-based requirements engineering can help to overcome some of the difficulties related to natural language requirements (e.g. ambiguity, confusion of abstraction layers or perspectives).

A conceptual model is an abstraction of a real world partition that is based on a conceptual modeling language (cf. [13]). Conceptual models help to *separate requirements specifications from design specifications*. Design constraints and design choices can be documented in separate models, e.g. by using an architecture description language. Models can be *checked automatically*, e.g. for consistency between different views or in order to identify the absence of certain requirements (cf. [14]). Conceptual models *support communication* among the stakeholders and help to acquire a common understanding of the planned system (cf. [16]). Additionally, conceptual models *support adhering to specific abstractions layers and system perspectives* by providing a restricted set of modelling elements and permitted relationships (cf. [56]).

2.3.2 *Continuous modelling in REMsES*

In order to combine the abstraction layers and content categories, we have selected a number of well-established modelling techniques and adapted or extended them to support consistent modelling across the three abstraction layers (see Sect. 2.1.2). These techniques are: context modelling (e.g. [30, 31]), goal modelling (e.g. [11, 36]), scenario modelling (e.g. [32, 45]), function modelling (e.g. [49]), architecture modelling (e.g. [39]), and behavior modelling (e.g. [23]). Figure 3 shows the assignment of these modelling techniques to the abstraction layers and content categories.

The modelling techniques provide mechanisms to ensure the consistency of models between abstraction layers. Each modelling technique can be applied across the three abstraction layers and, while some of them stick to one content category, others explicitly belong to two categories. For instance, scenario modelling is assigned to the requirements category, expressing that scenario models should be used preferentially for documenting requirements. In contrast, goal modelling is assigned to two categories, namely context and requirements. Modelling techniques that belong to two content categories, facilitate the integration of the corresponding content categories (e.g. goal modelling integrates the content categories “context” and “requirements”). In the following, the modelling techniques that have been adapted to be used in the approach are described at a glance:

– *Context Modelling* develops a model of the environment of a planned system, a function group, or a software or hardware component, respectively. We distinguish between business context, stakeholder context, and operational context (cf. [55]). For instance, a context model of the operational context represents, among other things, actors who use the system or component and environment variables that the system must control.
– *Goal Modelling* provides an overview of the intentions of the stakeholders together with the corresponding characteristic functional and quality properties of the system, a function group, or a software component. Goals justify detailed requirements and design decisions but are defined independently from a specific technical solution. High-level goals are refined hierarchically into subgoals (cf. e.g. [34]).
– *Scenario Modelling* develops a scenario-based description of the system, a function group, or a software component in terms of sequences of interaction that exemplify the satisfaction of individual goals (cf. [44]). The most common technique for scenario-based descriptions are use cases [32]. Since scenarios describe, among other things, the satisfaction of individual stakeholders’ goals,

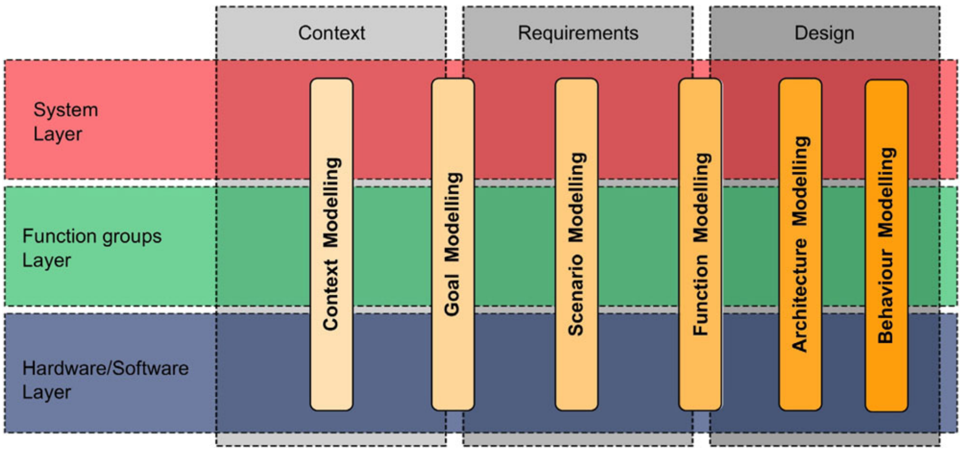


Fig. 3 Structure of the reference model with assigned specification techniques

goal models facilitate the integration of the content categories "context" and "requirements".

– *Function Modelling* documents the functions of the planned system and their relationships. A function model consists of an overview diagram of the functions and their communication relationships and template-based definitions of the individual functions (cf. [20]). Functions on the system layer are usage functions, functions on the function groups layer are logical functions, and functions on the hardware/software layer are technically implemented functions. An example for a relationship on the system layer is a usage dependency, on the function group layer it is a data flow, and on the hardware/software layer it is a call-relationship.
– *Architecture Modelling* defines the essential structures of the planned system in terms of components, connectors, and interfaces (cf. e.g. [39]). Since specific characteristics of the solution (system, function group) are relevant for specifying the requirements of the respective abstraction layer below, the architecture modelling in the approach is restricted to structural dependencies. This is due to co-design in the embedded systems domain, where, for example, the design of the overall system restricts the requirements on the function groups layer.
– *Behavior Modelling* defines the essential behavioral view, for example the state-based view of the system operations within the content category "design" (cf. [12]). This allows for a comprehensive specification of system and component behavior which is relevant for the specification of requirements of the system layer below.

How continuous modelling addresses the challenges Each modelling technique is restricted to a specific perspective that is relevant for supporting the specification of the system, a function group, a software component or a hardware component. Consequently, the resulting specification models only exhibit information that is relevant from that specific perspective. Statements across content categories can be made on the basis of cross-cutting language concepts. For instance, context models and scenario models contain the cross-cutting language concept of an "actor". The integration of the context and scenario model with respect to a specific actor permits to make statements that integrate information from these two content categories.

2.4 Focus on artefacts

The objective of this principle is to establish artefacts as the central concept of the approach.

2.4.1 *Artefact-centered approaches in general*

An *artefact* within a requirements engineering process can be characterized as a cohesive set of documented information. An *artefact type* defines joint properties of a set of similar artefacts. Within an artefact-based approach, a *document* is a coarse-grained logical grouping of artefacts and addresses a specific purpose (e.g. the "Customer Requirements Document" includes all information concerning the needs and constraints of the customers). A document type is defined by the set of artefact types which are used to document the information related to the specific document type.

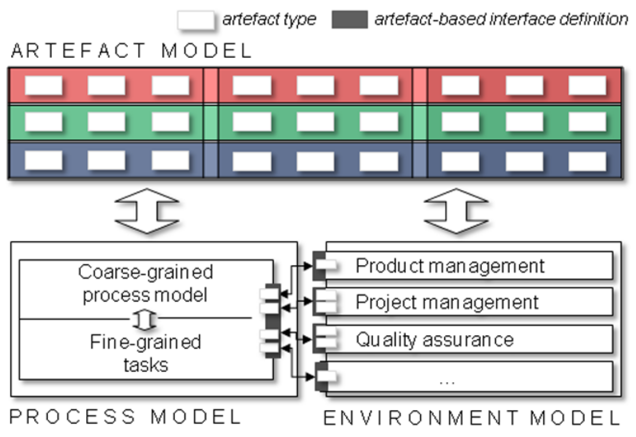


Fig. 4 The three defining elements of the approach

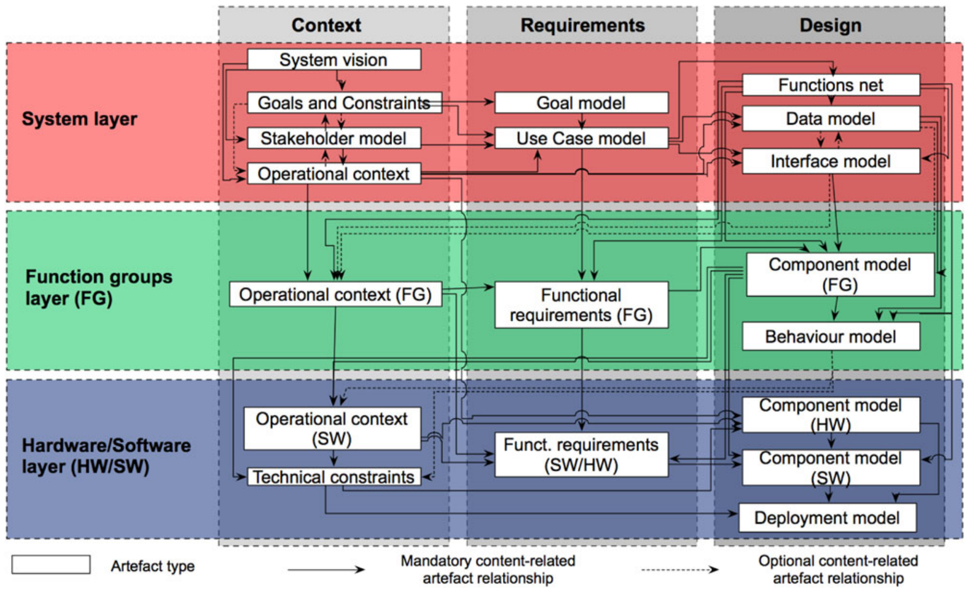
2.4.2 *Artefact-centering in REMsES*

The approach is defined by three interrelated models which are in turn based on the artefacts (see Fig. 4). This means that both the Process Model and the Environment Model are tightly connected to the Artefact Model as depicted in the figure.

The *Artefact Model* provides a basic structure for the definition of the artefacts, their assignment to abstraction layers and content categories, and the relations between the artefacts. The *Process Model* defines the coarse-grained course of action and fine-grained task descriptions. The former defines general control flow dependencies within requirements engineering processes. The latter model defines individual artefact-related tasks. Each of these tasks provides a structured description for supporting the systematic development of artefacts of the specific type. The interfaces between tasks or the execution of a task are defined on the basis of the development and completion of certain artefacts. The *Environment Model* defines the interfaces between the environmental processes that interact during the engineering process of the system with its requirements engineering process, for example, product management, project management, or quality assurance. These interfaces are defined in an artefact-oriented manner by demanding certain artefacts to be received, delivered, or exchanged.

By choosing a particular content category, abstraction layer, and specification technique, the modeler obtains a detailed characterization of the model to be created. However,

Fig. 5 Artefact types of the Artefact Model



the guidance for creating the model comes from three different sources: the descriptions of the corresponding content categories, the descriptions of the corresponding abstraction layers, and the descriptions of the corresponding modelling techniques.

Having to put together the guiding information from the three sources may overwhelm the modeler. Thus, we provide predefined *artefact type descriptions* for the different options concerning content category, abstraction layer, and specification technique. Figure 5 shows an overview of the artefact model and its content dependencies.

All artefacts derived from the Artefact Model are described using a template that, for example, includes the classification into corresponding content categories, abstraction layers, and specification techniques. In most cases, artefact descriptions suggest alternative notations, so that the developer can choose, for instance, the most suitable modelling tool. It should be noted, that in a specific project, an important step is to tailor the model and choose the artefacts that are relevant for the project rather than creating all artefacts defined by our model.

How the artefact-centered approach addresses the challenges Focusing on artefacts helps to cope with the number, size, and complexity of software-intensive systems in automotive industry. The Artefact Model structures the requirements engineering process by defining which information should be gathered and documented. The artefact-based interface definition between REMsES-based processes and interacting processes helps to integrate the approach in existing process landscapes. Additionally, the consideration of individual artefact types supports the specific validation and the assignment of responsibilities.

3 Realization of the REMsES guide

The four principles of the approach, which were introduced in Sect. 2, represent the foundation of the guideline. Each principle induces specific characteristics of the REMsES Guide.[5] In order to give an example, the principle *differentiation between abstraction layers* leads to a classification of artefacts within the guideline in terms of their assignment to the system layer, the function groups layer, and the hardware/software layer.

3.1 Structure of the guide

The guide was realized as a hypertext-based system. On the highest level the guide consists of the more specific *Process Guide* and the *Demonstrator*. Using the example of a "Radio-Frequency-Warning-System"[6] the Demonstrator shows the various artefacts that will be created during a requirements engineering process for a software-intensive embedded system. In the following, we focus on the first component of the guide, the Process Guide. Its overall content structure is shown in Fig. 6.

The Process Guide consists of seven sections:

[5]The REMsES Guide and further material, including extensive illustrating examples, is available free of charge at http://www.remses.org. This source also includes the complete list of project-related publications which complement the paper at hand, for instance, concerning details of the six REMsES modelling techniques. We decided not to present example instances of artefact descriptions within the article at hand, but we encourage the interested reader to download it.

[6]The Radio-Frequency-Warning-System (RFW) is a vehicle system that detects the radio-signature of road signs. Depending on the type of detected road signs and the driving direction the system initiates specific actions, e.g. the system informs the driver about a legal speed limit.

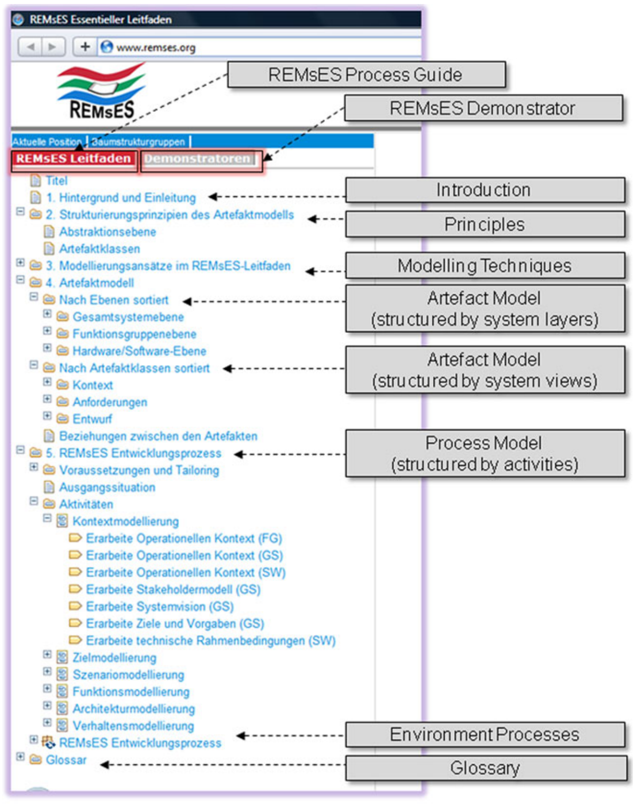

REMsES Essentieller Leitfaden
www.remses.org
REMsES
REMsES Process Guide
REMsES Demonstrator
Aktuelle Position | Baumstrukturgruppen |
REMsES Leitfaden
Demonstratoren
Titel
1. Hintergrund und Einleitung
2. Strukturierungsprinzipien des Artefaktmodells
Abstraktionsebene
Artefaktklassen
3. Modellierungsansätze im REMsES-Leitfaden
4. Artefaktmodell
Nach Ebenen sortiert
Gesamtsystemebene
Funktionsgruppenebene
Hardware/Software-Ebene
Nach Artefaktklassen sortiert
Kontext
Anforderungen
Entwurf
Beziehungen zwischen den Artefakten
5. REMsES Entwicklungsprozess
Voraussetzungen und Tailoring
Ausgangssituation
Aktivitäten
Kontextmodellierung
Erarbeite Operationellen Kontext (FG)
Erarbeite Operationellen Kontext (GS)
Erarbeite Operationellen Kontext (SW)
Erarbeite Stakeholdermodell (GS)
Erarbeite Systemvision (GS)
Erarbeite Ziele und Vorgaben (GS)
Erarbeite technische Rahmenbedingungen (SW)
Zielmodellierung
Szenariomodellierung
Funktionsmodellierung
Architekturmodellierung
Verhaltensmodellierung
REMsES Entwicklungsprozess
Glossar
Introduction
Principles
Modelling Techniques
Artefact Model (structured by system layers)
Artefact Model (structured by system views)
Process Model (structured by activities)
Environment Processes
Glossary

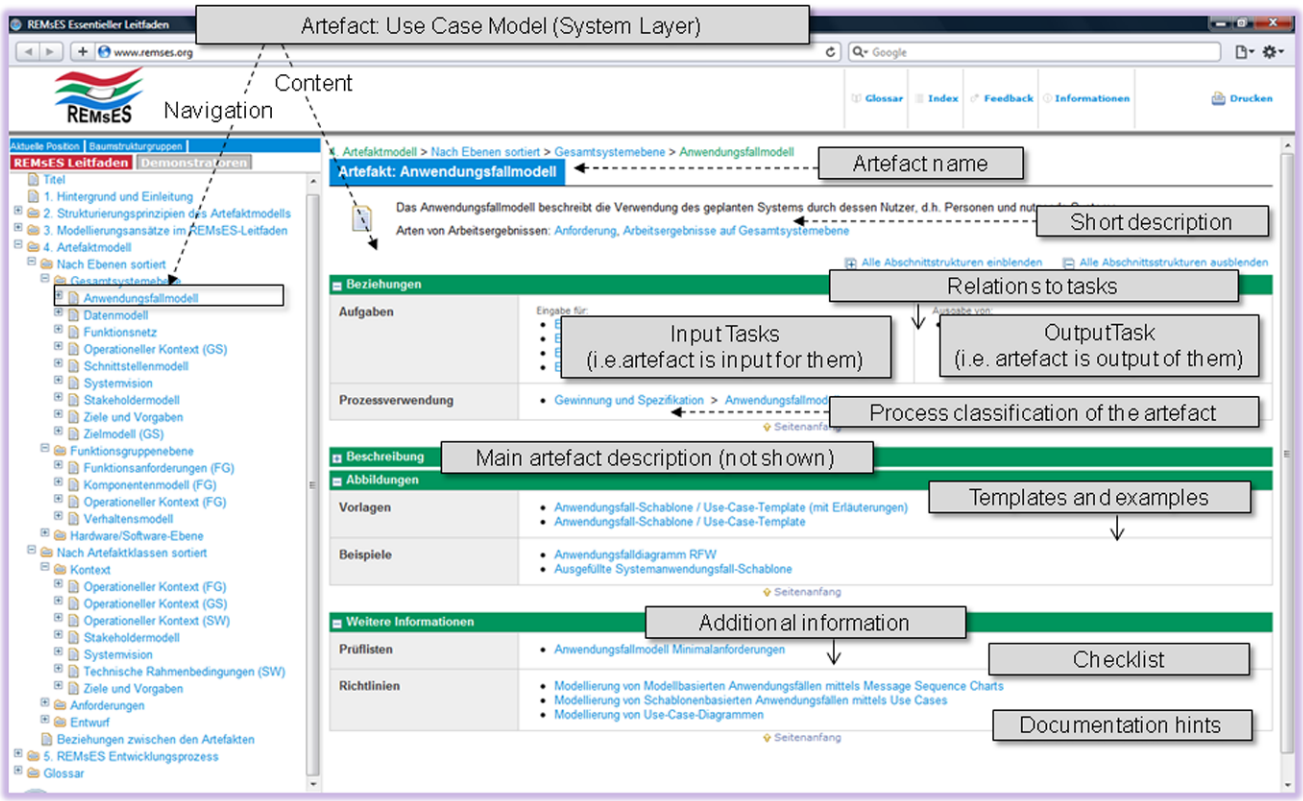
Fig. 6 Structure of the REMsES navigation frame

- *Introduction:* This chapter describes the problem statement and the vision of the approach.
- *Principles:* This chapter describes the core principles of the approach which were already presented in Sect. 2.
- *Modelling Technique:* This chapter refers to the corresponding modelling technique of the approach which were also presented in Sect 2.
- *Artefact Model:* This chapter describes the artefact model of the approach consisting of a detailed description of the individual artefact types. The artefact model within the guide is structured in two different ways. The user of the guide can choose one of these structuring mechanisms as an entry point for accessing the artefact descriptions within the guide. The artefact description within the artefact model is classified by the three system layers or by the three different system views (see Sect. 2). Section 3.2 presents the content structure of the artefact descriptions.
- *Process Model:* This chapter describes the Process Model consisting of fine-grained task descriptions for systematically creating the individual artefacts. The structure of the task descriptions is presented in Sect. 3.3.
- *Environment Processes:* This chapter describes the interfaces between REMsES processes and environmental processes that provide information for requirements engineering (e.g. product management) or processes that use information created during requirements engineering. The interface between REMsES processes and the environmental processes (e.g. product management, project management, or quality assurance) is specified in terms of artefact and corresponding artefact flows.
- *Glossary:* The Glossary defines the technical terms that are used in the approach. The glossary contains more than 50 definitions of technical terms.

3.2 Process guide: artefact descriptions

The Artefact Model defines artefact types and the relationships between them with regard to their content. Artefact descriptions in the process guide provide detailed information that is relevant for the documentation and quality assurance of the concerning artefact. The structure of artefact descriptions within the guide is illustrated in Fig. 7.

- *Artefact name* contains the name of the artefact type. If artefacts of a specific kind occur on more than one system layer, the name of the layer is added to the artefact name, e.g. "Operational context (FG)".
- *Short description* contains a brief description of the purpose of the artefact type. This description includes, among other things, the rationale why artefacts of this type have to be created and why they are relevant in requirements engineering processes.
- *Relationships to tasks* contains the various relationships between artefacts of this type and the tasks within the Process Model. The tasks are differentiated into input tasks (tasks that require artefacts of the specific type as input) and output tasks (tasks that create artefacts of the specific type).
- *Process classification of the artefact* explains, to which coarse-grained requirements engineering activity the artefact can be assigned (e.g. elicitation, specification).
- *Main artefact description* refines the short description of the artefact and describes the detailed structure of the content of the artefact (e.g. the individual slots of a use case description or the types of elements in a context model). Complementary to the abstract content structure and semantics, appropriate documentation techniques are suggested (e.g. textual documentation or model-based documentation).
- *Templates and examples* provides templates for structured textual documentation or reference models, depending on the suggested documentation techniques. Besides, this section contains examples for artefacts of the corresponding artefact type.
- *Additional information* contains evaluation criteria for checking the quality of the created artefact and rules that define when sufficient information with regard to an artefact has been documented and, thus, when the description of the artefact is complete.


Artefact: Use Case Model (System Layer)
Content
Navigation
REMsES Essentieller Leitfaden
www.remses.org
Glossar
Index
Feedback
Informationen
Drucken
Aktuelle Position
Baumstrukturgruppen
REMsES Leitfaden
Demonstratoren
Titel
1. Hintergrund und Einleitung
2. Strukturierungsprinzipien des Artefaktmodells
3. Modellierungsansätze im REMsES-Leitfaden
4. Artefaktmodell
Nach Ebenen sortiert
Gesamtsystemebene
Anwendungsfallmodell
Datenmodell
Funktionsnetz
Operationeller Kontext (GS)
Schnittstellenmodell
Systemvision
Stakeholdermodell
Ziele und Vorgaben
Zielmodell (GS)
Funktionsgruppenebene
Funktionsanforderungen (FG)
Komponentenmodell (FG)
Operationeller Kontext (FG)
Verhaltensmodell
Hardware/Software-Ebene
Nach Artefaktklassen sortiert
Kontext
Operationeller Kontext (FG)
Operationeller Kontext (GS)
Operationeller Kontext (SW)
Stakeholdermodell
Systemvision
Technische Rahmenbedingungen (SW)
Ziele und Vorgaben
Anforderungen
Entwurf
Beziehungen zwischen den Artefakten
5. REMsES Entwicklungsprozess
Glossar
Artefact name
Short description
Relations to tasks
Input Tasks (i.e.artefact is input for them)
OutputTask (i.e. artefact is output of them)
Process classification of the artefact
Main artefact description (not shown)
Templates and examples
Additional information
Checklist
Documentation hints
Alle Abschnittstrukturen einblenden
Alle Abschnittsstrukturen ausblenden
Seitenanfang

4. Artefaktmodell > Nach Ebenen sortiert > Gesamtsystemebene > Anwendungsfallmodell

Artefakt: Anwendungsfallmodell

Das Anwendungsfallmodell beschreibt die Verwendung des geplanten Systems durch dessen Nutzer, d.h. Personen und nut

Arten von Arbeitsergebnissen: Anforderung, Arbeitsergebnisse auf Gesamtsystemebene

Beziehungen

Aufgaben	Eingabe für: Ausgabe von:
Prozessverwendung	• Gewinnung und Spezifikation > Anwendungsfallmod

Beschreibung

Abbildungen

Vorlagen	• Anwendungsfall-Schablone / Use-Case-Template (mit Erläuterungen) • Anwendungsfall-Schablone / Use-Case-Template
Beispiele	• Anwendungsfalldiagramm RFW • Ausgefüllte Systemanwendungsfall-Schablone

Weitere Informationen

Prüflisten	• Anwendungsfallmodell Minimalanforderungen
Richtlinien	• Modellierung von Modellbasierten Anwendungsfällen mittels Message Sequence Charts • Modellierung von Schablonenbasierten Anwendungsfällen mittels Use Cases • Modellierung von Use-Case-Diagrammen

Fig. 7 Structure of an artefact description

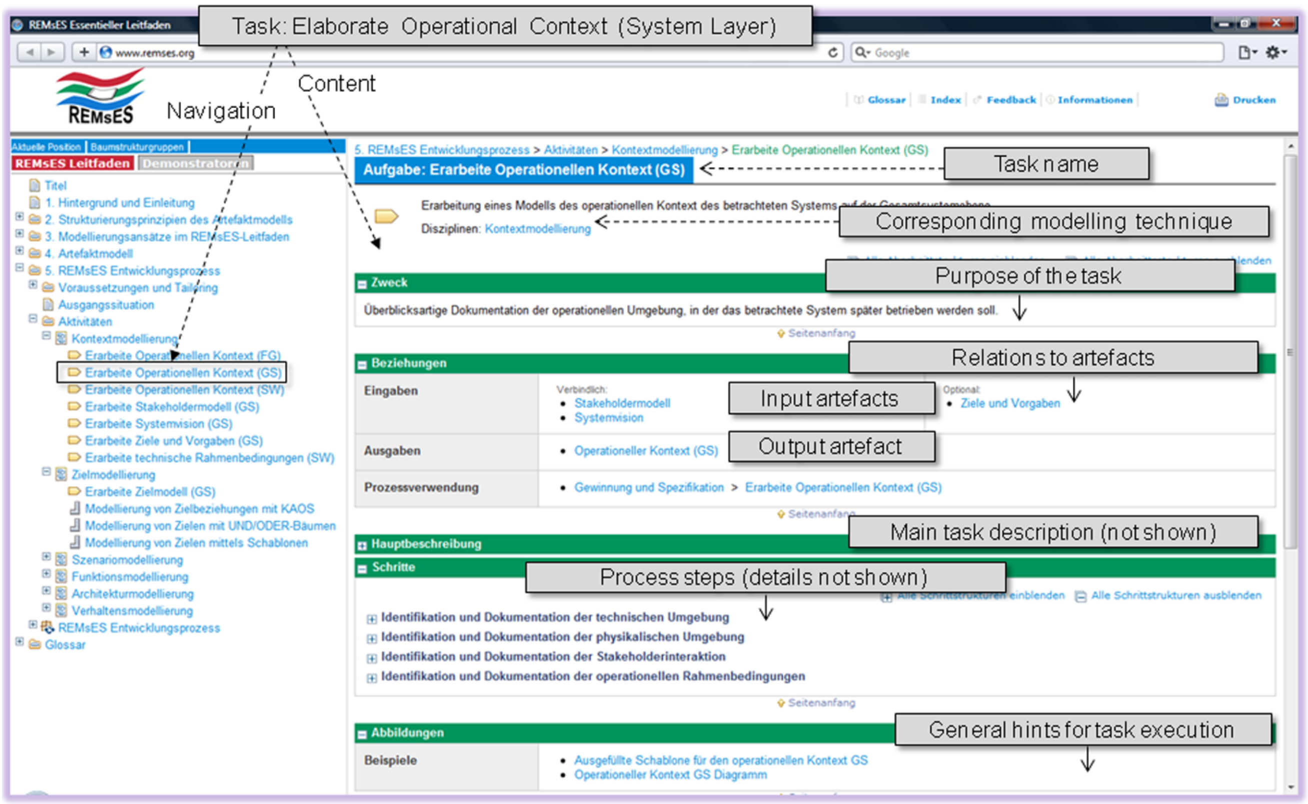
3.3 Process guide: task descriptions

The Process Model consists of a coarse-grained and a fine-grained process model (see Sect. 2.4). The structure of a task description within the fine-grained process model is illustrated in Fig. 8.

– *Task name* contains the name of the tasks. Typically, the tasks name consists of the activity followed by the name of the specific artefact type ("Document Technical Constraint"). If tasks of a specific kind occur on more than one abstraction layer the name of the layer is added to the tasks name, e.g. "Document Operational context (FG)".
– *Corresponding modelling technique* shows the classification of the task to an individual modelling technique, described in Sect. 2.3.
– *Purpose of the task* contains a description of the intention of the task, e.g. the systematic gathering and consolidation of the required information together with an appropriate documentation of the information.
– *Relations to artefacts* describes relations of the task to artefacts of specific artefact types. The related artefacts are differentiated as input artefacts and output artefacts. Input artefacts of a task are artefacts that are mandatory or optionally required to perform that task. The output artefact indicates the artefact type that is instantiated by performing the task, i.e. the type of the artefact that is created by the task.
– *Task main description* provides a detailed description of the task. The main description details the purpose of the task together with a general specification how the input artefacts are to be used in order to create the output artefacts.
– *Process steps* provides individual process steps that should be executed to develop the artefacts of the specific artefact type systematically. The process steps also describe how the input information (i.e. input artefacts) should be analyzed and how the gathered information should be consolidated and enriched to create the output artefacts. Whenever necessary, each of the process steps gives hints on how to elicit missing artefact information.
– *General hints for task execution* provides hints for supporting task execution, e.g. stakeholders who should be involved or documents that should be considered when eliciting missing artefact information.

3.4 Tailoring requirements engineering processes in automotive industry

In order to apply the guide in a specific process context, the guideline has to be tailored. For this purpose the relevant


Task: Elaborate Operational Context (System Layer)
Navigation
Content
Task name
Corresponding modelling technique
Purpose of the task
Relations to artefacts
Input artefacts
Output artefact
Main task description (not shown)
Process steps (details not shown)
General hints for task execution
REMsES Essentieller Leitfaden
www.remses.org
Google
Glossar | Index | Feedback | Informationen
Drucken
Aktuelle Position | Baumstrukturgruppen
REMsES Leitfaden
Demonstratoren
Seitenanfang

Titel
1. Hintergrund und Einleitung
2. Strukturierungsprinzipien des Artefaktmodells
3. Modellierungsansätze im REMsES-Leitfaden
4. Artefaktmodell
5. REMsES Entwicklungsprozess
Voraussetzungen und Tailoring
Ausgangssituation
Aktivitäten
Kontextmodellierung
Erarbeite Operationellen Kontext (FG)
Erarbeite Operationellen Kontext (GS)
Erarbeite Operationellen Kontext (SW)
Erarbeite Stakeholdermodell (GS)
Erarbeite Systemvision (GS)
Erarbeite Ziele und Vorgaben (GS)
Erarbeite technische Rahmenbedingungen (SW)
Zielmodellierung
Erarbeite Zielmodell (GS)
Modellierung von Zielbeziehungen mit KAOS
Modellierung von Zielen mit UND/ODER-Bäumen
Modellierung von Zielen mittels Schablonen
Szenariomodellierung
Funktionsmodellierung
Architekturmodellierung
Verhaltensmodellierung
REMsES Entwicklungsprozess
Glossar

5. REMsES Entwicklungsprozess > Aktivitäten > Kontextmodellierung > Erarbeite Operationellen Kontext (GS)

Aufgabe: Erarbeite Operationellen Kontext (GS)

Erarbeitung eines Modells des operationellen Kontext des betrachteten Systems auf der Gesamtsystemebene

Disziplinen: Kontextmodellierung

Zweck

Überblicksartige Dokumentation der operationellen Umgebung, in der das betrachtete System später betrieben werden soll.

Beziehungen

Eingaben	Verbindlich: • Stakeholdermodell • Systemvision	Optional: • Ziele und Vorgaben
Ausgaben	• Operationeller Kontext (GS)	
Prozessverwendung	• Gewinnung und Spezifikation > Erarbeite Operationellen Kontext (GS)	

Hauptbeschreibung

Schritte

Alle Schrittstrukturen einblenden   Alle Schrittstrukturen ausblenden

- **Identifikation und Dokumentation der technischen Umgebung**
- **Identifikation und Dokumentation der physikalischen Umgebung**
- **Identifikation und Dokumentation der Stakeholderinteraktion**
- **Identifikation und Dokumentation der operationellen Rahmenbedingungen**

Abbildungen

Beispiele	• Ausgefüllte Schablone für den operationellen Kontext GS • Operationeller Kontext GS Diagramm

Fig. 8 Structure of a task description

artefact types have to be determined and roles from the overall development process together with responsibilities have to be assigned to artefact types. The artefact-based definition of the approach provides a basis for tailoring requirements engineering processes by selecting relevant artefact types and their assignment to specific roles in engineering process. In particular, this holds for the manifold occurrences of supplier-manufacturer-relationships which are typical in automotive industry. However, the tailoring process for the guide was not in the scope of the project.

4 Tool support for REMsES processes

Besides the documentation of the Artefact Model and Process Model (see Sect. 3.1), a developer needs specific support for developing and managing artefacts. A variety of tools exists in the area of requirements management and model-based design that are suitable for developing the individual artefacts of the Artefact Model (Sect. 2.4) and that are already widely established in industrial practice. Examples for such tools include a range of UML tools for modelling use cases and activity diagrams, depicting functional requirements and behavior models, or MSC editors for scenarios. Available products are, for instance, IBM Telelogic Doors,[7] Borland Together,[8] or SparxSystems Enterprise Architect.[9]

The range of available tools gives rise to the necessity to use a dedicated repository to manage the artefacts and their relations according to the Artefact Model. So we developed a concept and a tool prototype for the management of artefacts and their relations. Thereby, the types of relations between artefacts are grouped into structural dependencies, consistency relations, and generative relations:

– *Structural dependencies* between artefacts are, in the field of requirements engineering, mainly composition relations ("A is part of B").
– *Consistency relations* realize dependencies which can be checked (contrary to structural dependencies). Depending on the artefact types and the dependency, the corresponding checks can be executed automatically or have to be performed manually. An example for a consistency relation states, that an artefact that describes a component and another artefact that describes the behavior of this component are interface-compatible with each other. Other ex-

[7] www-01.ibm.com/software/awdtools/doors/.
[8] www.borland.com/de/products/together/index.html.
[9] www.sparxsystems.com.au/products/ea/index.html.

amples are dependencies describing refinement or design rationales.
- *Generative relations* describe that (and how) artefacts are generated automatically from other artefacts. Thus, if a new version of a source artefact is available, a follow-up generation process has to be triggered. Generative relations declare, for example, that a test protocol was generated from a specific behavior model and a specific (test) scenario description.

4.1 Challenges for the tool support

Our industrial project partners emphasized the following challenges:

- In accordance with the principle "Focus on artefacts" (see Sect. 2.4) a set of artefacts has to be managed. The number of artefacts is variable and there are different types and formats of artefacts. This does also include additional artefacts that are not necessarily listed in the artefact model, e.g. meeting protocols or presentations. Furthermore, an artefact may be represented by a document or may be a part of a document containing many artefacts.
- There are relations and dependencies between artefacts, which have to be managed. Relations have different types and can be parameterized. For example, consistency relations can be parameterized with the name of the role in the process that is responsible for proving the consistency.
- The interconnectedness of the artefacts raises the need for consistency checks, thereby supporting seamless model-based development and adhering to principle "Distinguish between problem and solution" (see Sect. 2.3). This is impaired by the fact that the artefacts do not necessarily build on a common information model. In part, the consistency checks can be executed automatically, but typically the majority is performed manually by developers.
- Apart from artefacts and their relations, there are validity states that have to be managed. The state of an artefact can be "under *work*", "*check content* required", "*check relation* required", and "*finished*". The state of a relation can be "*check* required" and "*valid*".
- Assembling artefacts requires version controlling. This includes versioning the artefacts themselves, the relations, and the states of artefacts and relations.
- The artefacts require well-defined and well-supported change management that accompanies all change processes and supports the developer in transitioning the artefact model from one consistent state to another consistent state.

The aim of providing a tool support for the REMsES processes is to present a light-weight tooling concept based on a framework for a generic workflow that supports the development of an adaptable artefact model (see Sect. 2.4).

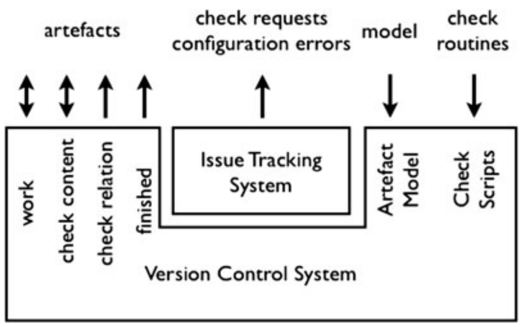
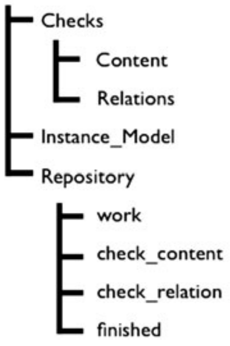


Fig. 9 Architectural Concept of the tool

Fig. 10 Structure of the artefact repository



This framework can be instantiated with an adapted artefact model and an adapted workflow. The workflow thereby specifies the order and detailed content of the checks that are performed.

4.2 Architectural concept

To meet the challenges discussed above, we developed a logical and technical tool architecture that provides the requested support, as shown in Fig. 9. It is based on a version control system (VCS), with a specific structure, which is complemented by an issue tracking system.

The VCS serves as a central database. The issue tracking system provides the ability to integrate human beings into the (validation) workflow. Thereby, the generic workflow is enforced by executing checks, restricting check-ins, and issuing tickets.

Some parts of the workflow cannot be executed automatically as it is not possible to express many of the test criteria in formal terms. As a consequence, manual consistency checks have to be integrated. We achieve this through the combination of the VCS with the issue tracking system, in a way that manual check tasks are issued as tickets enforcing a workflow.

Repository structure The VCS keeps track of the different versions of each document containing artefacts and the current state of the document. Figure 10 shows the structure of the artefact repository. The directory "repository" contains the artefacts, which are filed in their respective subdirectories, depending on their current state. For the user, only the directories "work" and "check content" are writable. "check relation" and "finished" are maintained by the system. "work" provides a space, where developers can save their daily work without any consequences, as this directory does not excerpt any effects if changes are committed. The generic workflow starts once the user commits a document to the directory "check content". This incorporates that the predefined checks are executed. As the kind of checks, which have to be performed, depend on the contained artefacts of the document, the type of each contained artefact has to be provided as meta-data of the document (as defined in the artefact model).

The directory "relations" holds the currently valid relations stored in the instance model. The implementation of the check scripts for content-based checks and dependency checks can be found in the "checks" directory.

Content and relation checks Content and relation checks can both be executed automatically and manually. Every check is triggered by invoking a shell script and returns "pass" or "fail". After the invocation, in case of a manual test, the script creates a ticket for a predefined role, which has to perform the check. The corresponding ticket can be closed with result "pass" or "fail". In case of a failure, the author of the document is informed via a ticket. For automatic checks, the procedure is the same, with the exception of automatic execution of a specific check script.

The specific implementation of the checks has to be defined for each type of artefact. To cope with this situation, our concept foresees organizing the checks as part of the repository. As a consequence, this eases the adaption of the system to different project properties and allows for redefining the checks.

4.3 Prototype

We realized the proposed concept on the basis of the two well-known and widely used, freely available tools: Subversion[10] as version control system and Trac[11] as issue tracking system. In this section we describe the integrated tools and the functionality of the prototype.

Subversion Subversion provides the functionality for versioning documents. User specific extensions could be implemented with so called hook scripts. There exist different kinds of hook scripts, which are executed either before a commit starts (pre-commit) or after a commit is finished (post-commit). Within the prototype post-commit scripts realize the checks. These scripts evaluate the meta-data, which describe the types of the contained artefacts, and invoke the corresponding checks. Possible results are:

– *pass*, if the check can be performed instantaneously and it is passed.
– *fail*, if the check can be performed instantaneously and it is not passed.
– *check pending* means that the check was not performed. This is basically the case if a ticket is issued, which is not closed immediately. In this case the system has to keep track of the state of the document.

Trac Trac is mainly used as a ticket system. It allows to create tickets automatically and to modify them with user defined properties. In our scenario, trac integrates the developers in the way that the check scripts create tickets for specific roles, e.g. quality assurance manager. After the reviewer has performed a check, the corresponding ticket can be closed and trac returns the result to the VCS.

Realization of the instance model The instance model stores the dependencies between artefacts and is implemented using the functionality of the eclipse modelling framework[12] (EMF). The project specific artefact model is modelled as an ecore-model. By making use of the EMF code generation facility, we generated a software component, which allows creating and accessing the corresponding instance model. It also provides the VCS with the information, which checks have to be performed in the different states of an artefact. Figure 11 shows a screenshot of the tool used to maintain the instance model of the demonstrator, the 'Radio-Frequency-Warning-System" case study (see Sect. 3.1).

Implementing checks The check routines and the dependency information are also managed via the VCS. Upon every check event, the system searches the relevant path in the repository for corresponding check scripts.

Functionality We illustrate the functionality of our prototype by describing the exemplary workflow of the use case artefact "Display warning to driver".

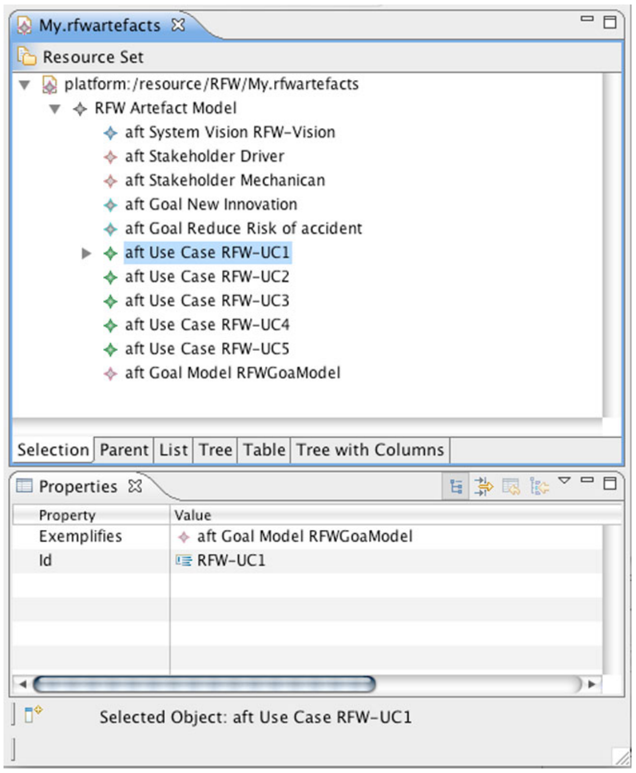
If the author commits the document to "check content",[13] a ticket to the quality manager of the project is issued. Subversion invokes a pre-commit hook, which ensures that the REMsES-specific meta-data exist, i.e. that the artefact is a use case and that an artefact of this type can be added to the

[10]www.subversion.org.

[11]trac.edgewall.com.

[12]www.eclipse.org/emf.

[13]For example, subversion could maintain the history of a document by using svn move and svn copy.


My.rfwartefacts

Resource Set

platform:/resource/RFW/My.rfwartefacts
- RFW Artefact Model
  - aft System Vision RFW-Vision
  - aft Stakeholder Driver
  - aft Stakeholder Mechanican
  - aft Goal New Innovation
  - aft Goal Reduce Risk of accident
  - aft Use Case RFW-UC1
  - aft Use Case RFW-UC2
  - aft Use Case RFW-UC3
  - aft Use Case RFW-UC4
  - aft Use Case RFW-UC5
  - aft Goal Model RFWGoaModel

Selection | Parent | List | Tree | Table | Tree with Columns

Properties

Property	Value
Exemplifies	aft Goal Model RFWGoaModel
Id	RFW-UC1

Selected Object: aft Use Case RFW-UC1

Fig. 11 RFW Instance Model

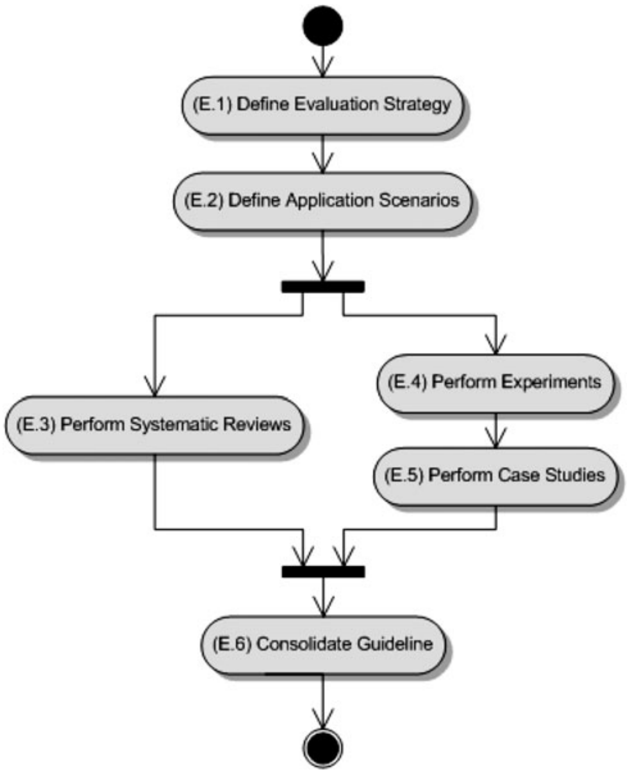

(E.1) Define Evaluation Strategy
(E.2) Define Application Scenarios
(E.3) Perform Systematic Reviews
(E.4) Perform Experiments
(E.5) Perform Case Studies
(E.6) Consolidate Guideline

Fig. 12 Evaluation strategy

instance model. After the commit was successful, the post-commit hook triggers an automatic check of the content, for example to ensure that the correct template has been used. Afterwards, the project manager is informed (via ticket), that the instance model needs to be updated. In our example, this means that he or she has to specify which goal is exemplified by this use case. This information is needed for the VCS to complete the state "check relation".

If the check fails, a ticket is issued to the owner of the document. The owner has to revise the document in "work" before he or she submits the revised version to be checked once more by committing it to "check content". After the check was successful, the manual content check is triggered by issuing a ticket for example to the quality assurance manager. If he closes the ticket with status "pass", the state of the document is changed to "check relation".

Similarly, the check of the relations could be processed. If all relation checks are passed the document is finally transferred into the directory "finished", which is equivalent to state "finished".

Assessment of the prototype The implementation shows that the proposed concepts can be realized by means of freely available tools, which must be slightly modified. The prototype was evaluated using the "Radio-Frequency-Warning-System" case study. The overall evaluation of the approach is described in the next section.

5 Evaluation of the REMsES guide

In accordance with [6, 53], the guide was evaluated by demonstrating its applicability and usefulness in the considered application area. The evaluation of the guideline was conducted within two strands. First, the guideline was reviewed by domain experts and process experts. Secondly, the guideline was evaluated by means of experiments and case studies. This section presents the major findings of the evaluation process.[14]

5.1 Evaluation process

The evaluation process of the guide is based on systematic reviews with requirements engineering experts and developers from the industrial partners as well as the application of selected parts of the guideline in experiments and case studies. As shown in Fig. 12 the evaluation strategy consists of several steps.

First of all, the shown evaluation strategy itself was defined (E.1). Secondly, application scenarios for selected parts of the guideline were defined (E.2). Therefore, our industrial partners prepared specifications for two software-intensive embedded systems in the automotive domain to

[14]Details concerning the evaluation activities have already been published in [37] and [40].

demonstrate the feasibility of the approach, namely the "Radio-Frequency-Warning-System" and the "Door-Lock-System".

5.1.1 Gap analysis and reviews

The evaluation by review comprised a systematic gap analysis and reviews (E.3). Several developers and process experts from various business units reviewed the guideline in workshops and guided interviews.[15] They compared REMsES with the existing processes to understand how the approach could fit into their development processes. The continuous reviews (every 3 to 6 months) allowed for a comprehensive evaluation of the guideline and provided valuable improvement suggestions. Findings from these reviews were used to consolidate the guideline (E.6).

5.1.2 Experiments and case studies

Beside gap analysis and reviews, two experiments were conducted with student participants. The experiments focused on selected parts of the guide in order to evaluate specific parts of the guideline (E.4). The first experiment was conducted with 12 students at Ulm University (cf. [37]). Additionally, an exploratory comparison to a state-of-the-practice process (SotP) was included. The students were grouped in six teams, four using the guideline and two groups followed the SotP. To achieve a better quality of the results two different systems had to be developed (called "MachZu", a controller for a central locking system and "Lumiere", a controller for a light system).

In the second experiment 11 students from University of Kaiserslautern worked in two groups on the specification of a door lock system using the two modelling techniques goal and scenario modelling across the three abstraction layers (cf. [40]). All students were recruited through bulletin and participated voluntarily. The students were naive to the goal of the experiment.

Based on the experiences of the experiments, two larger case studies at the University of Applied Sciences in Esslingen were conducted to evaluate how development projects can benefit from systematic goal/scenario modelling (E.5). This study incorporated a full development project with a complete development life-cycle. The students had to understand a customer specification of a door lock system, create a system requirements specification, implement the system using the programming language "C", deploy it, and finally test the realized system.

In the first study, students were split into 6 groups consisting of 5 to 6 students each. All groups participated in the same lecture on requirements engineering for embedded software. Three of them were additionally trained to use the goal/scenario approach and they were told to apply it. The other three groups were allowed to specify the requirements using techniques from the general lecture. We called this the "ad-hoc" approach, although the students were trained at techniques which are typically used in companies. For every group, key metrics were collected in a tracking system. These metrics measured the effort in every development phase and the number of successfully executed test cases at the end of each project. Over the course of the case study, the groups were interviewed. The second case study was a replication of the first case study with an increased number of groups and thus an increased number of samples. This time 55 students were divided into 10 groups. Students were recruited in the same manner described above, naive to the nature of the investigation, and participated voluntarily.

The empirical results were analyzed to get deeper evidences concerning the applicability and usefulness of the approach. In addition findings from the evaluation activities were used for revisions of the guide (E.6).

5.2 Empirical results

The evaluation activities provide strong evidences for the applicability and usefulness of the guide by improving the quality of specifications without an additional development effort. The experiment [37] at the University of Ulm had two major results concerning the comprehensibility of the guideline and the quality of specification. In this context comprehensibility can be seen as a facet of applicability.

5.2.1 Comprehensibility of the guide

The first result shows that the guide is suitable at least in the application area of the case studies. Based on interviews of the students, the artefacts and process steps were rated with a median score of 3, which means "well understandable" (see Fig. 13). Furthermore the process steps seem to guide the development of each artefact well.

5.2.2 Specification quality

The second result of the experiment demonstrates that the specification quality significantly increases. For this purpose a checklist consisting of categories such as correctness or consistency was used. Figure 14 shows the quality score of the specification of each group. A MannWhitney U test with an U value of 0.1 gives significant evidence that the specification quality of the approach is better than the quality of the SotP approach.

The second experiment confirms the results of the first experiment. Furthermore, it indicates that by applying a use

[15] Workshops and guided reviews were chosen over statistically significant questionnaires as we wanted concrete feedback on applicability and possible improvements rather than a rating.

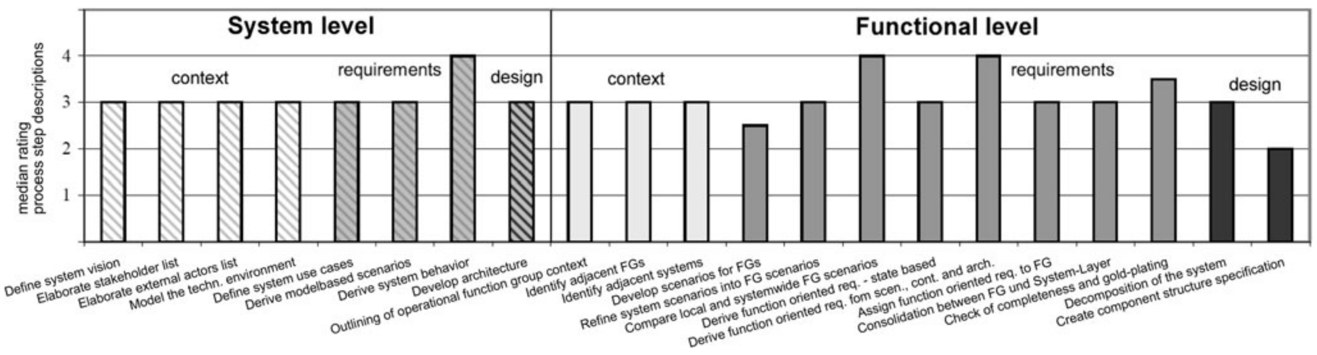


Fig. 13 Median rating with a range from 1 (worst) to 4 (best) of the process step descriptions

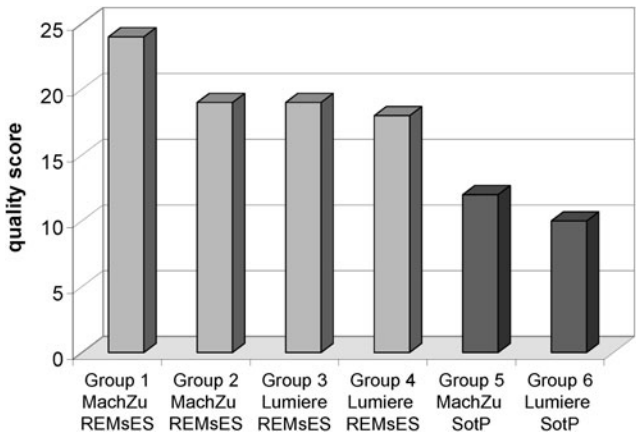


Fig. 14 Checklist based specification quality scores

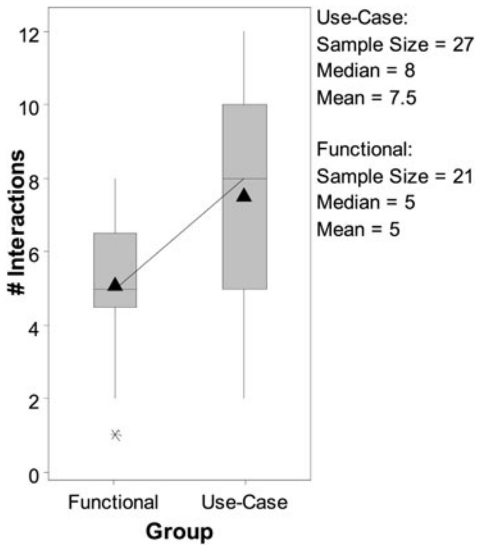


Fig. 15 Described interaction in experiment

case-based approach (e.g. goal modelling and scenario modelling) significantly more interactions were described, as depicted in Fig. 15 (cf. [40]). This figure shows boxplots of the number of interactions between components being described in the different specifications. On average (based on the median), the groups that were using the use case approach specified 3 interactions more than the remaining groups per artefact.

5.2.3 Development effort

Additionally, the two case studies give strong evidence that the approach is cost-effective and reduces the effort variance. Initially, we expected "better" requirements at higher expense of development effort. Contrary to our expectations, this increase did not occur. As depicted in Fig. 16 the application of goal/scenario models in the two case studies did not significantly increase the overall project effort.

Figure 16 shows two boxplots of the total efforts for the student groups and the individual values. As it can be seen, the median and mean values do not differ much, but the variance in total effort is higher for the ad-hoc groups compared to the goal/scenario groups.

This effect can by traced back to variances during the coding phase. The ad-hoc groups had a much higher variance of their effort during the coding phase than the goal/scenario groups. This is shown in Fig. 17. In summary, there are authoritative indications that the approach can be applied in a cost-effective way. On the other hand, with respect to quality improvement, no differences in terms of successfully executed test cases could by observed. Both groups passed roughly the same amount of test cases.

5.2.4 Threats to validity

From our point of view, students are not fully representative of professionals. Therefore, we believe that a generalization of the absolute numbers is not possible. Furthermore, the number of case studies and experiments is too low to have statistically significant results with respect to development efforts and quality.

We think that the comparison between the goal/scenario technique and the "ad-hoc" approach does not represent a

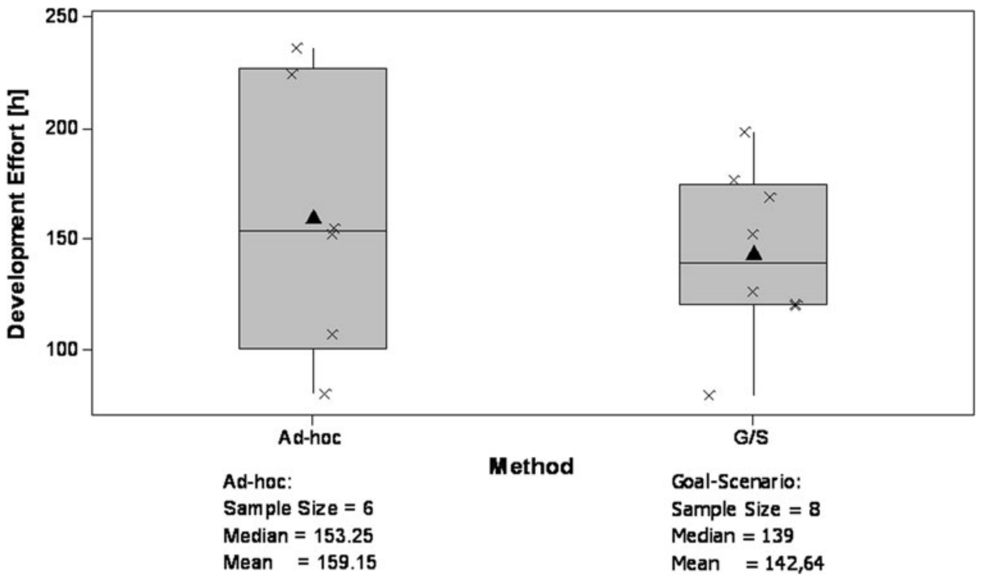


Fig. 16 Overall development effort in the case studies

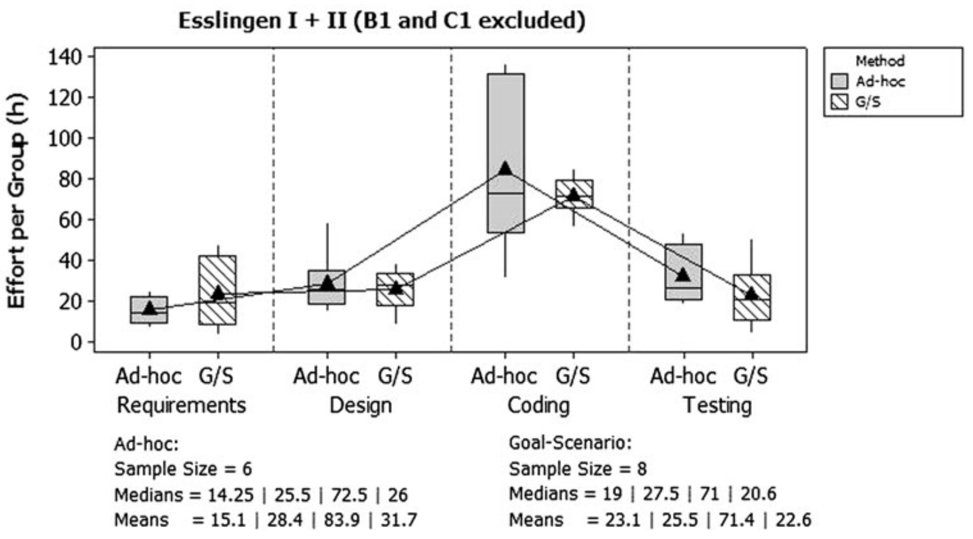


Fig. 17 Effort per phase in the case studies

crucial threat to validity. Although the term "ad hoc" implies that the students had no special guidance, they applied techniques they knew from the lecture. These techniques are techniques that are frequently used in companies. We validated our assumption (i.e. that the students use techniques which are commonly used in industrial practice) by comparing the requirements documents created by the students to requirements documents from the companies.

5.2.5 *Conclusions from evaluation*

In summary, there are strong indications to assume that the REMsES approach may have a positive impact if used appropriately. Compared to industrial automotive systems, the experimental tasks have a rather small to medium size. Therefore we believe that our empirical studies show trends, but further empirical evidence is needed before these results can be generalized for real complex systems development.

6 Related work

The following section discusses the work that is directly related with the project or concerned with a closely related topic.

The book by Robertson and Robertson [48] concerning the topic requirements engineering is wide-spread throughout the industry. The authors provide systematic guidance with practical insights, techniques, and templates for supporting requirements engineering processes. In contrast to the processes guidance for requirements engineering proposed in this and other approaches, REMsES is artefact-centred and thereby considers the specific characteristics of software-intensive embedded systems (e.g. the differentiation between software and hardware).

The IEEE Recommended Practice for Software Requirements Specifications [27] recommends a structure and outline for requirements documents, and the IEEE Guide for Developing System Requirements Specifications [25] provides for a respective outline for system requirements documents. Both recommendations do not include further instructions on methods or modelling techniques.

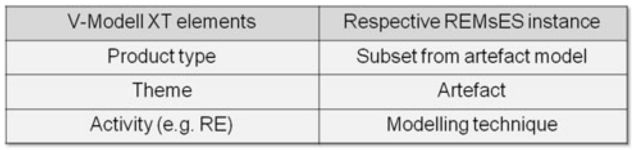
In [4] a reference model for requirements engineering for embedded systems has been proposed with an artefact scheme. Though, this work does not recommend or give a detailed elaboration of the artefacts' contents or specific modelling techniques. [4] served as an input for the development of the REMsES artefact model.

Approaches from the automotive industry often emphasize on architecture description languages (ADLs). These can be used for some of the proposed artefacts (i.e. the component models) in the content category "Design" of the artefact model. For example the AUTOSAR consortium proposes a standard for architectural descriptions [24] that is intended to be used by original equipment manufacturers as well as suppliers to ease distributed development and integration. At about the same time, the project EAST-EEA proposed the EAST ADL [38], but after its initial release, the architecture description language has not been developed any further. In [42] the authors proposed an experience-based approach for integrating architecture and requirements engineering, but without providing a concrete artefact structure.

Focused rather on the design and the technical architecture, the project ARTEMIS proposes to use their Rich Components approach [9], while the design environment METROPOLIS [1] supports a formal platform-based design method for hardware and software co-design based on formal semantics. The project AutoMoDe [15] also proposes a model-based design approach with the help of transformations. Each of these approaches could be integrated with the REMsES approach, thereby providing for a smooth transition to the design phase.

Concerning process support, the ISO provides an process assessment model [28], while Chrissie et al. give guidelines for process integration and product improvement for CMMI [7]. Another well-known process model is the Rational Unified Process (RUP) framework [33] that provides best practices for software and systems development. Instead of competing with available process support approaches, REMsES differs by emphasizing an artefact-based approach to requirements engineering that does not depend on a particular development process but works in combination with any of them.

Especially for the automotive domain, the SPICE consortium offers Automotive SPICE, composed by a process assessment model [50] and a process reference model [51].



V-Modell XT elements	Respective REMsES instance
Product type	Subset from artefact model
Theme	Artefact
Activity (e.g. RE)	Modelling technique

Fig. 18 Mapping REMsES contents to the V-Modell XT

The process support in these works considers what information has to be captured, yet is hardly concerned with how contents can be captured or processed. In contrast, REMsES provides concrete guidance on the single steps that have to be taken and embeds the development process explicitly into surrounding processes.

Integration with one specific standard process framework, namely the V-Modell XT [5], has already been performed exemplarily for the REMsES Guide. The V-Modell XT is a framework for the definition of process models that is well-known and widely applied in the industry in large-scale projects. The general advancement when developing a process comprises three main steps: First, define a process model by choosing activities from the V-Modell XT. Secondly, select the type of project (e.g. supplier- manufacturer project for embedded systems). Thirdly, develop a project execution strategy of how the chosen activities are to be performed. The third step includes the integration of the guide for the requirements engineering activities chosen in the first step, as depicted in Fig. 18.

Thereby, the product types of the V-Modell XT are mapped to artefact subsets from the artefact model (see Fig. 5). The V-Modell XT product types are hierarchically decomposed into themes which can consequently be mapped to single artefacts. On this basis, the integration into the respective process phases can be derived.

7 Conclusion and outlook

The approach aims at supporting requirements engineering processes for software-intensive embedded systems in the automotive industry. In this paper, we introduce the four fundamental principles of the approach and give a structural overview over the guide. Furthermore, we presented the existing tool support, the evaluation process and the findings which we gain from the evaluation activities.

7.1 Contribution to the state-of-the-art

The presented approach is based on existing modelling techniques. These techniques are indeed already used in industrial practice, but, to a large extent, they are used in isolation and in an ad-hoc manner. Based on their expert knowledge, engineers have to decide in this situation which available information should be used to develop the new artefact and how the characteristics of the new artefact are influenced by other artefacts.

Our approach contributes to the state-of-the-art by adapting these techniques with respect to the four REMsES principles. For example, each modelling technique has been adapted to support the consistent documentation of information across the three abstraction layers.

The artefact model of the approach yields the integration of artefacts and corresponding modelling techniques by explicitly defining relationships between the content of different artefact types. Beyond that, our approach also facilitates the integration of modelling techniques by providing a systematic process guidance to create artefacts, consistently, across the three abstraction layers and by using information that is already documented within other artefacts.

The contribution of the approach is additionally increased by its tooling concept. Together with the guide, the tooling concept provides a concrete technical infrastructure for supporting requirements engineering processes in practice.

Even though we have not yet evaluated the guide in any large industrial setting, the feedback from the experiments and case studies we have performed and the existing tool support for the approach has been very encouraging. We found evidences that our approach can serve as a vehicle for improving the quality and cost-efficiency of requirements engineering processes in the automotive industry. Furthermore, regarding its application in industry, we observed that the approach constitutes an initial step towards a common requirements engineering terminology and the comparability of requirements engineering processes in industrial practice.

7.2 Ongoing and future work

The research project SPES (Software Platform Embedded Systems) which is currently a key project in German research on engineering theories for embedded systems aims at developing a holistic domain-independent development platform for engineering embedded systems. From the perspective of the SPES project the REMsES approach (i.e. principles, techniques, guideline, and tool support) is a significant domain-specific input. Currently, as part of the SPES project, the approach is applied and extended to serve a broader spectrum of domains (e.g. automation, avionic, energy, and medicine).

The next stage for bringing the approach into practice is to execute a pilot project in the industry based on the general process integration described by use of the V-Modell XT and then integrate the guide into the actual process of the respective company.

Acknowledgements We would like to thank the following persons for their valuable contributions throughout the project: Jörg Leuser from Daimler AG, Nadine Bramsiepe, Günter Halmans, Ernst Sikora, Bastian Tenbergen, and Nelufar Ulfat-Bunyadi from Universität Duisburg-Essen, Wassiou Sitou, Eva Geisberger, and Klaus Lochmann from Technische Universität München, Igor Menzel and Franz Grzeschniok from Bosch Corporate Research, Prof. Jörg Friedrich from the Esslingen University of Applied Science, and Markus Bechter and Jan Philipps from Validas AG. We would also like to thank the anonymous reviewers for their feedback.

References

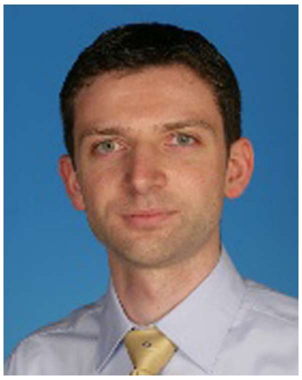
1. Balarin F, Passerone R, Pinto A, Sangiovanni-Vincentelli A (2005) A formal approach to system level design: metamodels and unified design environments. In: Proceedings of the 3rd ACM and IEEE int conf on formal methods and models for Co-Design. ACM Press, New York, pp 155–163
2. Broy M (2006a) Challenges in automotive software engineering. In: Osterweil LJ, Rombach HD, Soffa ML (eds) Proceedings of the 28th int conf on software engineering. ACM Press, New York, pp 33–42
3. Broy M (2006b) The 'grand challenge' in informatics: engineering software-intensive systems. IEEE Comput 39(10):72–80
4. Broy M, Geisberger E, Kazmeier J, Rudorfer A, Beetz K (2007) Ein Requirements-Engineering-Referenzmodell. Informatik-Spektrum 30:127–142
5. Bundesministerium des Inneren (2009) V-Modell XT. http://www.cio.bund.de/DE/IT-Methoden/V-Modell_XT/v-modell_xt_node.html
6. Cao J, Crews JM, Deokar A, Burgoon JK, Nunamaker JF (2006) Interactions between system evaluation and theory testing—a demonstration of the power of a multifaceted approach to information systems research. J Manag Inf Syst 22(4):207–235
7. Chrissis MB, Konrad M, Shrum S (2006) CMMI: guidelines for process integration and product improvement. The SEI series in software engineering. Addison-Wesley, Reading
8. Cockburn A (2000) Writing effective use cases. Addison-Wesley/Longman, Boston
9. Damm W, Votintseva A, Metzner A, Josko B, Peikenkamp T, Böde E (2005) Boosting re-use of embedded automotive applications through rich components. In: Proceedings of foundations of interface technologies. Elsevier, Amsterdam. Electronic Notes in Theoretical Computer Science
10. Dannenberg J, Kleinhans C (2004) The coming age of collaboration in the automotive industry. Mercer Manage J 18:88–94
11. Dardenne A, van Lamsweerde A, Fickas S (1993) Goal-directed requirements acquisition. Sci Comput Program 30(1/2):232–282
12. Davis AM (1993) Software requirements: objects, functions, and states. Prentice Hall, Upper Saddle River
13. Falkenberg ED, Hesse W, Lindgreen P, Nilsson BE, Jan Oei JL, Rolland C, Stamper RK, van Assche FJM, VerriJn-Stuart AA, Voss K (1998) A framework for information system concepts—the Frisco report. Tech. rep., International Federation for Information Processing (IFIP), Luxemburg
14. Finkelstein A, Gabbay D, Hunter A, Kramer J, Nuseibeh B (1994) Inconsistency handling in multi-perspective specifications. IEEE Trans Softw Eng 20(8):569–578
15. Freund U, Braun P, Romberg J, Bauer A, Mai P, Ziegenbein D (2006) Automode—a transformation based approach for the model-based design of embedded automotive software. In: Proceedings of the 3rd European congress on embedded real time software. SIA, Toulouse
16. Gause D, Weinberg G (1989) Exploring requirements—quality before design. Dorset House, New York
17. Ghezzi C, Jazayeri M, Mandrioli D (1991) Fundamentals of software engineering. Prentice Hall, Englewood

18. Gorschek T, Wohlin C (2006) Requirements abstraction model. Requir Eng 11(1):79–101
19. Grimm K (2003) Software technology in an automotive company—major challenges. In: Proceedings of the 25th int conf on software engineering. IEEE Computer Society, Washington, pp 498–503
20. Grünbauer J (2008) Feature Interactions auf Nutzungsebene. Softwaretechnik-Trends (1). Gesellschaft für Informatik, Bonn
21. Hammond J, Rawlings R, Hall A (2001) Will it work? In: Proceedings of the 5th IEEE int symp on requirements engineering. IEEE Computer Society, Washington, pp 102–109
22. Hardung B, Kölzow T, Krüger A (2004) Reuse of software in distributed embedded automotive systems. In: Proceedings of the 4th ACM int conf on embedded software. ACM, New York, pp 203–210
23. Harel D (1987) Statecharts—a visual formalism for complex systems. Sci Comput Program 8:231–274
24. Heinecke H, Schnelle KP, Fennel H, Bortolazzi J, Lundh L, Leflour J, Maté JL, Nishikawa K, Scharnhorst T (2004) Automotive open system architecture—an industry-wide initiative to manage the complexity of emerging automotive E/E architectures. Convergence Transportation Electronics Association, pp 325–332
25. Institute of Electric and Electronic Engineers (1998a) Guide for developing system requirements specifications (ANSI/IEEE Std 1233-1998)
26. Institute of Electric and Electronic Engineers (1998b) Guide for information technology—system definition—concept of operations (IEEE Std 1362-1998)
27. Institute of Electric and Electronic Engineers (1998c) Recommended practice for software requirements specifications (IEEE Std 830-1998)
28. International Organization for Standardization (2006) Information technology—process assessment. Part 5. An exemplar process assessment model (ISO/IEC Std 15504-5:2006)
29. Jackson M (1995a) Software requirements and specifications: a lexicon of practice, principles and prejudices. ACM Press/Addison-Wesley, New York
30. Jackson M (1995b) The world and the machine. In: Proceedings of the 17th int conf on software engineering. ACM, New York, pp 283–292
31. Jackson M (2001) Problem analysis and structure. In: Hoare T, Broy MRS (eds) Engineering theories of software construction. IOS Press, Amsterdam, pp 3–20
32. Jacobson I, Christerson M, Jonsson P, Oevergaard G (1992) Object oriented software engineering—a use case driven approach. Addison-Wesley, Reading
33. Kruchten P (2000) The rational unified process: an introduction. Addison-Wesley/Longman, Boston
34. van Lamsweerde A (2001) Goal-oriented requirements engineering: a guided tour. In: Proceedings of the 5th IEEE int symp on requirements engineering. IEEE Computer Society, Washington, p 249
35. van Lamsweerde A (2008) Requirements engineering: from craft to discipline. In: Proceeding of the 16th ACM SIGSOFT int symp on foundations of software engineering. ACM, New York, pp 238–249
36. van Lamsweerde A, Dardenne A, Delcourt B, Dubisy F (1991) The KAOS project—knowledge acquisition in automated specifications of software. In: Proceedings of AAAI spring symposium series. American Association for Artificial Intelligence, Standford, pp 69–82
37. Leuser J, Porta N, Bolz A, Raschke A (2009) Empirical validation of a requirements engineering process guide. In: 13th int conf on evaluation and assessment in software engineering
38. Lonn H, Saxena T, Nolin M, Torngren M (2004) FAR EAST: Modeling an automotive software architecture using the EAST ADL. In: Proceedings of the 1st int ICSE workshop on software engineering for automotive systems. IEEE Computer Society, Washington
39. Medvidovic N, Taylor R (2000) A classification and comparison framework for software architecture description languages. IEEE Trans Softw Eng 26(1):70–93
40. Menzel I (2009) Functional versus use-case based requirements specification: an experiment. Technical report, Robert Bosch GmbH, Stuttgart
41. Nuseibeh B (2001) Weaving together requirements and architectures. IEEE Comput 34(3):115–119
42. Paech B, von Knethen A, Doerr J, Bayer J, Kerkow D, Kolb R, Trendowicz A, Punter T, Dutoit A (ICSE 2003) An experience-based approach for integrating architecture and requirements engineering. In: 2nd int software requirements to architectures workshop
43. Parnas DL, Madey J (1995) Functional documents for computer systems. Sci Comput Program 25(1):41–61
44. Pohl K, Sikora E (2007) Structuring the co-design of requirements and architecture. In: Proceedings of the 13th int working conference requirements engineering-foundation for software quality. LNCS, vol 4542. Springer, Heidelberg, pp 48–62
45. Potts C (1995) Using schematic scenarios to understand user needs. In: Proceedings of the ACM symposium on designing interactive systems. ACM, New York, pp 247–266
46. Pretschner A, Broy M, Kruger I, Stauner T (2007) Software engineering for automotive systems: a roadmap. In: Proceeding of future of software engineering. IEEE Computer Society, Washington, pp 285–303
47. Rinke T, Weyer T (2007) Defining reference models for modelling qualities—how requirements engineering techniques can help. In: Proceedings of the 13th int working conference requirements engineering-foundation for software quality. LNCS, vol 4542. Springer, Heidelberg, pp 335–340
48. Robertson S, Robertson J (1999) Mastering the requirements process. Addison-Wesley, Reading
49. Ross DT, Schoman KE (1977) Structured analysis for requirements definition. IEEE Trans Sofw Eng 3(1):6–15
50. SPICE User Group (2008a) Automotive SPICE—process assessment model (PAM). http://www.automotivespice.com/
51. SPICE User Group (2008b) Automotive SPICE—Process reference model (PRM). http://www.automotivespice.com/
52. Swartout W, Balzer R (1982) On the inevitable interwining of specification and implementation. Commun ACM 25(7):438–440
53. Walls JG, Widmeyer GR, Sawy AE (1992) Building an information system design theory for vigilant eis. Inf Syst Res 3(1):36–59
54. Weber M, Weisbrod J (2003) Requirements engineering in automotive development: experiences and challenges. IEEE Softw 20(1):16–24
55. Weyer T, Pohl K (2008) Eine Referenzstrukturierung zur modellbasierten Kontextanalyse im Requirements Engineering softwareintensiver eingebetteter Systeme. In: Kühne T, Steimann F (eds) Tagungsband zur 'Modellierung 2008'. LNI, vol 127. Gesellschaft für Informatik, Bonn, pp 181–196
56. Wieringa RJ (2002) Design methods for software systems: yourdon, statemate and UML. Kaufmann, San Francisco
57. Zave P, Jackson M (1997) Four dark corners of requirements engineering. ACM Trans Softw Eng Methodol 6(1):1–30

Peter Braun received a diploma in computer science and a doctorate in computer science from Technische Universität München, Germany, in 1997 and 2004. He is co-founder of Validas AG and a member of the management board. He has expertise in model-based development, model transformations, testing, and in development processes especially for automotive embedded systems.

Manfred Broy is a full professor for computer science at the Technische Universität München. His research interests are software and systems engineering comprising both theoretical and practical aspects. This includes system models, specification and refinement of systems, development methods and verification techniques.

Frank Houdek is a Manager at Daimler AG, Research and Development, responsible for processes and methodologies in Requirements Engineering for all business divisions. He headed several Research and Advanced Engineering projects on Requirements Engineering and Management. Frank holds a Diploma and Ph.D. in Computer Science from University of Ulm (Germany). He is a member of IEEE CS and GI. He is a founding member of the IREB (International Requirements Engineering Board).



Matthias Kirchmayr is a requirements engineer at Daimler AG group research. His focus is on standardisation and optimisation of methods and processes of requirements engineering and management. Matthias received Ph.D. in natural science at the university of Ulm.

Mark Müller is currently a research engineer and project manager at Bosch Corporate Research since 2003. He had prior experience in software development and worked as a freelancer for small to medium enterprises. Since he joined Bosch, his major work areas comprise software process improvement and quality management, performance measurement, empirical methods and business analysis. Mark holds a diploma in electrical and computer engineering from the University of Hannover (Germany) and a doctorate from the University of Kaiserslautern (Germany).

Birgit Penzenstadler has been working as research associate at the Software & Systems Engineering group at the Technische Universität München since 2006. Her research interests are requirements engineering and system design. The topic of her dissertation is the transition of requirements from systems to subsystems.

Klaus Pohl is a full professor for Software Systems Engineering at the University of Duisburg-Essen and adjunct-professor at the University of Limerick, Ireland. His research interests include requirements engineering, service-based system engineering, software quality assurance and software product lines. Klaus Pohl is the coordinator of the FP7 Network of Excellence S-Cube (Software Services and Systems Network), vice-chair of the steering committee of the European Technology Platform, member of the NESSI executive board, and the steering committee of the German Innovation-Alliance SPES 2020 (Software Platform for Embedded Systems). He has served as program and general chair of more than 10 international conferences. He is (co)author of more than 130 refereed publications and several textbooks. Among other things, he served as independent expert in several expert working groups of the European Commission.

Thorsten Weyer leads a group of researchers focusing on requirements engineering and service engineering at "paluno—The Ruhr Institute for Software Technology" at University of Duisburg-Essen. Prior to this he was a research associate in the Software Systems Engineering group at the University of Duisburg-Essen. Thorsten Weyer was the coordinator of the REMsES project and is a supporting member of the International Requirements Engineering Board (IREB). He received a Diploma degree in computer science from University of Koblenz with distinction and is an experienced consultant and lecturer in the software engineering and requirements engineering area for more than ten years now. His dissertation deals with the analytical quality assurance of behavioral specifications against properties of the operational environment.