Reuseable Subsystems from an Overall System Specification

Birgit Penzenstadler
penzenst@in.tum.de
Technische Universität München
Fakultät für Informatik
Software & Systems Engineering
85748 Garching bei München, Germany

Abstract

Firstly, decomposition of multifunctional embedded systems into subsystems implies a long list of relevant criteria to be taken into consideration. Secondly, reuse of such subsystems is already done in practice but is often just performed for code or lacks a systematic approach. In this paper, we describe a method for defining and outlining subsystems from the overall system requirements and specification, then extracting a subsystem by tracing all its relevant artifacts, and finally we describe the steps for its reuse within a new surrounding system that is being developed.

1 Motivation

Within all the efforts to make software development more efficient, one of the leading paradigms is reuse. The vision is a construction kit for plugging together the components that have been developed for earlier systems or (for product lines) during domain engineering with as little configuration effort as possible for your newly to-be-developed products or (again for product lines) during application engineering. There are two major challenges for such a construction kit: One is to appropriately define the building blocks and the other is to facilitate their reuse. Thereby reuse shall not be limited to the implementation (the actual code) but incorporates all assets or artifacts that are relevant for the particular (sub)system. This helps to save effort not only during the implementation phase but already within the requirements engineering phase of the new system and throughout the whole development process. To achieve this, we need expertise on how to cut a system into potentially reusable subsystems.

1.1 OEM-supplier-relationships

The automotive domain is a prominent representative for the development of multifunctional embedded systems and will therefore serve as example in this paper. In their current product line development they sign up suppliers for many subsystems [36]. These OEM-supplier-relationships also influence the decomposition of the system [18] and have to be considered additionally to the functional, architectural, and quality constraints. This influence has not yet been addressed sufficiently in relation with the development of reusable subsystems.

1.2 Notions of Reuse

There are two notions of reuse in the area of product lines that imply different handling: One is reuse within the product line that was planned during domain engineering when setting up the product line [30] and the other one is unplanned reuse or post-development reuse. Although product lines focus primarily on planned reuse, we additionally mention unplanned or opportunistic reuse as it is for example relevant for promoting self-developed subsystems to components off-the-shelf (COTS).

Regarding the timeline of software development, we can also distinguish between development for reuse and development with reuse [20]. The development process for software product lines is usually divided into the subprocesses of domain and application engineering, including backflows and iterations. So during domain engineering the emphasis is on development for reuse, although you may already desire reuse here if you do not start the product line on greenfield but already have legacy products. During application engineering the emphasis is on development with reuse, but you might find a new feature requirement which leads to a
new component that may be worth being added to the common platform for reuse. Both stages can include the use of COTS.

**Development for Reuse:** Development for reuse will be concerned with the first steps of our method, the appropriate transition from systems in subsystems, with the intention of finding reusable subsystems. Artefacts developed for reuse often provide the possibility to customize them by configuration. Extreme configurability though may lead to problems as well, mainly a significant increase in effort when recombining the different variants to products, but the methodological question when development for reuse is adequate will not be discussed any further within the scope of this work.

**Development with Reuse:** Development with reuse will rely on the later steps of our method, the extraction of all relevant information and the subsystem’s integration into a new environment. This can include adaptation and modification of the to-be-reused subsystem which may require additional effort.

**Lack of systematic reuse:** As stated in [32] the problem is not a lack of software reuse, but a lack of systematic reuse. There is a lot of effort put into achieving reuse in the different notions explained above but there are is still a lack of systematic methods for achieving it.

### 1.3 Contribution

Our goal for the thesis is twofold: We want to investigate the criteria for “good” decomposition into subsystems for the domain of multifunctional embedded systems with OEM-supplier relationships and we want to facilitate the systematic reuse of such subsystems. Therefore we have to tightly integrate requirements engineering with systems development and work on maximum traceability between the different stages. Our approach provides a method for the systematic reuse of subsystems and their specifications. It consists of a transition from system specification to a subsystem specification, including an analysis of the criteria that are relevant to divide a system into subsystems, then a process to extract the desired subsystem from the originally surrounding system, and finally the reuse of the subsystem by customizing it and integrating it into a new surrounding system. The method is especially valuable for software product lines as within that paradigm the development usually already contains a systematic approach to planned reuse and therefore facilitates the application of our approach. Furthermore, additional effort can be saved through considering the additional aspect of variability within the execution of our process steps.

### 1.4 Outline

In Section 2 we explain the basis for our approach. Section 3 depicts the method: We describe the transition from a system specification to a subsystem specification, which includes an analysis of the criteria that are relevant to divide a system into subsystems (Section 3.3.1), and a process to extract the desired subsystem from the originally surrounding system (Section 3.3.2). Then follows the reuse of the subsystem by customizing it and integrating it into a new surrounding system (Section 3.3.3). In Section 4, we apply the approach to software product lines and discuss the additional aspect that variability implies for the method. A proof of concept will be implemented within the AutoFOCUS tool that is described in Section 5. Finally, we refer to related work (Section 6) and provides ideas for future work (Section 7).

### 2 Background

In this section, we briefly sketch our background with regard to the definition of a system, the views onto a system, the relations and tracing within a system, and the process of requirements engineering as basis of system specification.

#### 2.1 Definition of a System

A system is generally defined as an “integrated coherency of objects, processes, and parts, which is either provided by nature or produced by mankind (...)” [25]. More specific, software systems consist of the “entirety of programs and data that builds an executable unit on a computer system to solve a certain task” [7]. For the development of such a system, we use modeling techniques that rely on different abstraction layers.

#### 2.2 Abstraction Layers

The abstraction levels referred to in Section 2.1 allow us different views onto the same system with emphasis on certain aspects that have to be considered during development. We use three abstraction layers [19]: The **usage** layer shows the results of requirements engineering including the functional hierarchy...
with usage interfaces (blackbox view), the **logical architecture** layer models represent the structure and internal behaviour of the system (whitebox view), and the **technical architecture** layer adds the code and task models and deals with issues concerning the hardware realization. Over the abstraction layers, the focus shifts from the complete system to function groups to realization components, providing more details on each layer and moving from abstract and implementation-independent to concrete and technical. Similar abstraction layers are used by [16].

The abstraction layers focus on different types of (horizontal) relations and interactions between the their respective entities. Furthermore, the vertical relationships between the layers can be followed through the use of a tracing model.

### 2.3 Traceability

Requirements traceability is defined as “the ability to describe and follow the lifecycle of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases).” [14]. Building on that definition, Ramesh and Jarke [33] provide a simple metamodel of requirements traceability that defines six dimensions of traceability. These are: **What** kind of information is represented? **Who** are the stakeholders? **Where** is the information represented? **How** is it represented? **Why** does the conceptual object exist? **When** is the information captured or modified? These dimensions are used in a reference model for requirements traceability that performs the four tasks of requirements management, design allocation, compliance verification, and rationale management [33]. We have not implemented this reference model within the Requirements Engineering Reference Model on which we base our approach (see Section 2.5), but we intend to further investigate in that direction.

The cited definition and reference model describe the so-called **vertical** traceability. Apart from that it is important to consider the **horizontal** traceability, meaning the relationships between artifacts of the same layer and their respective elements, for example dependencies, interferences or any other connection they might have.

### 2.4 Automotive Domain

We have already motivated the automotive domain as prominent representative for the development of multifunctional embedded systems (see Section 1). They participating parties have to deal with different restrictions that OEM-supplier-relationships add to the development process. For example, currently only whole electronic control units (ECUs) are assigned to suppliers and the software architecture has to be structured according to the ECU hardware [18]. This will improve with the progress and propagation of AUTOSAR [3] and its methodology, as OEMs will then be able to specify the software of an ECU through AUTOSAR Software Components. This will also ease the realization of distributed functionality and subsystems that require high interaction with other subsystems as for example in driver assistance systems.

While automotive systems design still mainly relies on textual specifications in natural language [36], there are some approaches to introduce model-based systems engineering, for example [16] or [5]. For our approach, we consider it important to start with the requirements engineering phase of the development life cycle, therefore we base our theory on the Requirements Engineering Reference Model from [12].

### 2.5 Requirements Engineering Reference Model

The Requirements Engineering Reference Model (REM) [12] is a Requirements Engineering (RE) Framework with an artifact model at its centre. The artifacts (see Figure 1) are the work results of the RE activities in product or product-line development and have refinement relations and dependencies between them. The framework is completed by a role model for the responsibilities and a tailoring approach for the set-up. The integration of the development of business needs, requirements specification and system specification provides a comprehensive approach to goal-
and system-oriented requirements engineering that includes all participating stakeholders and their different views on the system. Examples for those stakeholders are marketing people, managers, system engineers, mechanics, and future users. Their perspectives on the system result from their different kinds of requirements which will all be integrated in the overall artifact model. REM was developed in the domain of embedded systems but is also applicable to other realms of systems and software engineering.

We chose REM as basis for our approach because it reflects the influences of different stakeholders on the system and includes all information that mirrors their respective needs and goals. As we want to explore the aspects of the OEM-supplier-relationships during development, this reference model seemed the most appropriate in being capable of capturing all relevant additional information.

**Business Needs** specify customer and strategic requirements, including product and business goals of the system development. It consists of the following artifacts: Business Objectives and Customer Requirements, System Vision, General Conditions and Scope & Limitations, ROI and Business Risk, and System Success Factors.

**Requirements Specification** contains the product functional and non-functional requirements. They are analyzed and modeled from the customer and user perspective and derived (and justified by) from the Business Needs. The artifacts are: Functional Analysis Models, Domain Model, Non-functional Requirements Model, and Acceptance Criteria.

**System Specification** contains a detailed definition of the functional system concept; the required behavior of the considered system and its integration into the overall system and environment. It defines constraints to the detailed design and realization of the system (software, hardware - electrical, mechanical). The artifacts include: User Interface Specification / User Documentation, Functional System Concept, External Interface Specification, Design Constraints, and System Test Criteria.

The document structure for a specific product development project is subject to tailoring and the organization's process definition, but for explaining our method we will define a limited instance of REM in Section 3.

### 3 Systematic Reuse of Subsystems

Before describing the method we propose for the reuse of subsystems, it is necessary to define what we understand as a subsystem and from which basis we start. A **subsystem** is a secondary or subordinate system, or in other words an “area within a system that itself shows the characteristics of a system” [25]. In our context of software systems development, the distinction between system and subsystem is taken according to the current scope. A subsystem is itself a system but will be called a subsystem whenever we regard the surrounding system at the same time. For the systematic reuse of a subsystem, we do not only want to reuse the implementation but all artifacts (regarding business needs, requirements and system specification) that have been created during development (see Section 1).
The starting point for our method is therefore an artifact model that is an instance of REM.

3.1 Demands for the Artifact Model

REM is a framework that defines what information has to be specified and how it is interrelated. It does not explicitly prescribe any modelling techniques or notations for artifacts. For our specific example instance we request:

- Complete documentation of the system
- Documentation of the rationale
- Continuous integrated modelling
- Documented relationships between the artifacts with defined semantics
- Traceability: vertical (throughout development process) and horizontal (of dependencies)

We know that these requests are of general nature and will not automatically lead to our special designed artifact model. The intention for this listing is contrary: it shows that our approach will work for any other artifact model that provides the information required by REM and fulfills the demands presented here.

3.2 Artifact Model

The artifact model has been defined with the intention to provide a simple instance of REM for case study purposes. We do not consider it all-encompassing yet. It consists of the results of the requirements elicitation and analysis (see Figure 1) and the corresponding design and implementation. The model is structured according to the three abstraction layers described in Section 2.2.

- System usage: abstract of core functionality and main purpose, domain analysis, stakeholder analysis, use cases, scenarios, goals, and a decision model for variability
- Logical architecture: feature diagram, specification with rationale, functionality, structure, and behaviour
- Technical architecture: documentation of the realization with rationale, interaction, structure, and execution

The proposed artifacts include all information of Kruchten’s 4+1 view model on software architecture [23]: the use case view can be found within the system usage layer, logical view and process view are represented in the logical architecture, and deployment view and physical view are incorporated in the technical architecture. The decision model for variability on the system usage level shall ease the variability management as described in [37], because a simple feature model is not sufficient, as explained in [9]. Part of the logical and technical architecture resemble the specification and realization of the KobrA component model [1].

A subsystem is again a system that can be described separately with the only difference being the smaller contextual scope, as if zooming in on a part of the overall system. Therefore we can assume an equivalent set of artifacts for each subsystem.

The artifacts are documented textually and in UML2 [17] and the respective elements are linked horizontally and vertically (forward and backward with respect to the development process) via a tracing mechanism. The traces are directed and typed as horizontal for a dependency or vertical for a development.

3.3 Transition Procedure

The transition from systems to subsystems incorporates the two steps of dividing a system into subsystems and then relating all the relevant artifacts and information belonging to the specified subsystem. Then the extracted subsystem can be reused. An overview of our steps can be viewed in Figure 2 and they will be described in detail in the following.

Figure 3. Transition from Systems to Subsystems.

The transition is depicted in Figure 3 where the small boxes indicate the parts that belong to one specific subsystem.
The next section describes how to divide a system into subsystems considering aspects from different stakeholders.

3.3.1 Division into Subsystems

The first step is to define the relevant criteria for the division into subsystems. We have found four categories of criteria from different stakeholders: Functional, architectural, rights, and quality requirements.

Not all categories have to be present for a system in the same quantity, for example, a web application would probably have to conform to certain business rules that are not of any interest for an embedded system. The type of the specified system will determine to what extent each of them is relevant for its division into subsystems.

- **Functional criteria**: Functional clustering according to user perception, Functional dependencies, Functional interactions
- **Architectural criteria**: Communication requirements, Technical constraints, Design rules
- **Rights constraints**: Laws and standards, Licensing/patents, Information politics and business rules, Implications from OEM-supplier-relationships
- **Quality requirements**: Performance, reliability, maintainability, . . . , and Costs

All these constraints have to be considered during the process of requirements engineering. The user’s point of view provides the functional criteria. The architectural criteria derive from the constraints given by the technical solution domain and the future system environment as well as from architecture design rules. Prominent examples are for such guidelines or paradigms were presented by Parnas (concerning information hiding, [27] [28]) and Dijkstra (concerning structure, [11] [21]). The stakeholders and the quantity of the rights constraints will vary depending on the type of system — apart from laws and standards there can be licensing constraints, busines rules and influences from OEM-supplier-relationships.

Furthermore, for systematic reuse the analysis must identify variations to anticipate changes and the design must be chosen for adaptability [10]. Therefore another criterion for the decomposition of a system is the stability of the requirements or how fast they might change — the buzzword here is “software aging” [29]. As this can be regarded as an aspect of evolvability within the quality criteria, we did not put it on the list separately.

Due to the aforementioned dependency of the criteria on the type of system and its specific situation, we cannot give a general rule on how to incorporate all the criteria equally into the decision for the division into subsystems. The relevance of the criteria has to be evaluated separately for each system and according to that relevance, the division can be decided individually. We do not want to limit the general applicability of our method but we are forced to assume a certain type of system in the following to be able to justify our selection of criteria for the division. The catalogue of criteria will be the same but their priorities might vary for specific systems or types of systems. Therefore we assume an embedded system, for example one from the automotive domain. The most suitable abstraction layer for cutting out a subsystem seems to be the functional layer as it integrates the analysed user requirements with a coarse sketch of the system specification. From that starting point on we can extract the desired subsystem. The correspondent part from the REM artifact model that seems most appropriate to begin with is the functional analysis model in the requirements specification.

The **process steps** that have to be performed for the division into subsystems are:

- Review the reference catalogue of criteria and analyse their relevance for the specific system
- Gather possibly new criteria that are additionally relevant for the specific system
- Analyse relations and influences between all the criteria
- Assign them priorities or weights
- Divide system according to priorities and solve conflicts

We are currently working on a case study in the automotive embedded systems domain to validate the method and be able to illustrate the steps with concrete examples. The study makes use of the above presented REM framework and its artifacts as a demonstrative basis (see Section 2).

3.3.2 Extraction of Desired Subsystem

The main **process steps** to get a subsystem “ready for reuse” are the following:

- Define exact borders of the desired subsystems
- Associate all corresponding constraints and relevant information
check completeness within the extracted information.

When decided which specification artifacts are the most appropriate ones for the desired reuse, we have to get all the corresponding information about the specific subsystem by following the traces in both directions upwards to the most abstract requirements concerning the context and downwards to the implementation in soft- and hardware. For example, if we decide to start from a functional analysis model as suggested in Section 3.3.1, we would extract the part of the model to be reused, presumably a function cluster or a component, and capture the relationships to other entities via their interfaces to the chosen entity.

The next step is to associate the corresponding information from the rest of the artifacts. The overall system artifacts will contain a lot more information that is of no relevance for the extracted subsystem. Therefore, we suggest to adapt the related artifacts for reuse by leaving out the irrelevant information. According to the extent of dispensable information we will simply save a reduced version of the same artifact.

Applied to the REM artifact model, the extracted subsystem specification will contain the directly related parts of the quality requirements, the acceptance criteria as well as a subset of the domain model (all from the block “requirements analysis”), the refinement of that analysis that is documented in the system specification, and a small subset of each category of the relevant business needs.

After that, we need to validate the consistency within the extracted subsystem specification and check whether the extracted information is conform to the defined constraints of the different categories defined in Section 3.3.1. The result of this phase is a consolidated and consistent subsystem specification.

3.3.3 Reuse of a Subsystem

For the actual reuse of a subsystem as depicted in Figure 4 there are four process steps for validation and integration with the new surrounding system that have to be performed:

- Analyse additional constraints and requirements for the subsystem
- Calculate the efforts for their integration
- Integrate requirements and check consistency
- Integrate subsystem and specification into new system

In a new project or for a new product, we have a similar phase of requirements engineering that was already performed for the earlier developed systems, including the ones that produced the subsystems we want to reuse in the new to-be-developed system. This time though, we integrate the artifacts of the already developed subsystem from the start and reuse them.

The first step is now to check whether the requirements that led to the proposed subsystem reuse correspond to its extracted specification. In case there are additional requirements that were not present in the former surrounding system (which is likely), we have to analyse whether and how they can be satisfied using the given subsystem. If that is not possible, we have to calculate the amount of work to modify the subsystem and then decide whether it is an appropriate solution. This trade-off cannot be given in general terms but must be evaluated for each individual case, therefore we assume that in our case the subsystem fits the given requirements.

Next step is to check the additional constraints given by the new surrounding system and to integrate them with the former ones from the subsystem specification. When the specification is finally consistent again after performing required extensions or adaptations, we can integrate the subsystem with all its artifacts into the new surrounding system with its developing set of artifacts and continue its evolution as in usual systems development.

4 Integration with Product Lines

In this section we explain the benefit that our method brings to product line development, the additional aspect of variability that product lines bring with them, and the tailoring that is required for applying our method.

4.1 Added Value

Product lines can add a lot of value to your software development. The advantages include reduction of de-
velopment costs (time to market, maintenance costs), better coping with complexity (optimized cost prediction, organized evolution), and higher quality of the systems.

These benefits are accomplished through definition of the commonalities, better configurability for an easier derivation of products, faster development of variants thanks to a common domain engineering, parallel development of variants, more efficient testing through reuse, and maximum reuse through real modularity.

For the described approach, the most interesting point is a high degree of reuse. Our method facilitates the both notions of reuse mentioned in Section 1, the development for reuse during domain engineering (division into subsystems and extracting them) and the development with reuse during application engineering (integration into a partly new surrounding system).

4.2 Variability Aspects

The variability in software product lines adds two aspects to the issue that make the task of reuse of a subsystem more complex. The first aspect is the variability of the new surrounding system and the second one is the variability of the proposed subsystem itself. For both aspects, the three steps preparing for and enabling reuse are extended to analysing, checking and integrating all possible variants of both sides. However, this complexity might be reduced again through assurances that are valid for whole variation points and not only for one particular variant.

The precondition for such warranties is well-planned design in the direction of modularity, which means that the main effort during product line design lies rather on the branching process while creating the product line than on the joining part of the configuration process.

4.3 Application

The method’s application requires the tailoring according to the type of development and considering the given variabilities. Firstly, depending on the desired kind of reuse, we either concentrate on the first or on the second part of the method, or we apply the whole method to achieve its maximum benefit. And secondly, depending on the contained variability, we add the necessary iterations for each variant to the corresponding steps of division, extraction and/or integration.

It is not necessary to change the method steps themselves, they can simply be integrated into the product line development process. During domain engineering, we prepare the division into subsystems by analyzing the criteria from the very start of the requirements engineering phase to include all relevant aspects in our decision.

After completing a first iteration of the domain engineering and having developed a first release of the common platform, we are able to extract subsystems with their specific set of required artifacts. During domain engineering, we can reuse and integrate formerly extracted subsystems from the same or even from a different product line.

5 Tool Support

We envision to provide tool support for our method as a proof of concept and to facilitate its application. The tool AutoFOCUS2 [38] is a scientific research prototype that is an implementation based on the FOCUS approach [8] and which has been developed at Technische Universität München since 1995. AutoRAID (AutoFocus Requirement Analysis Integrating Development) is the extension for requirements engineering that has been added for the latest release.

AutoFOCUS2 is an integrated, extendable CASE prototype for a mathematically founded system specification and development tool. It includes a system concept with functional system views and graphical description techniques as well as test case and code generation, simulation and verification of components. AutoRAID adds the analysis and integration of requirements by first identifying and capturing the requirements, then refining and structuring them, and finally classifying and modelling them.

The next extension that is currently being planned with the development of AutoFOCUS3 is a product line add-on called AutoPLACE (AutoFocus Product Line Analysis and Construction Extension). In a first iteration we add variability to the AutoFOCUS modelling part and in a second iteration to the AutoRAID analysis part.

Finally we will add AutoREUSE as support for the definition and extraction of subsystems and their integration into newly defined systems. The user will be able to export a subsystem via an interactive dialogue where he can choose the desired artifacts according to the guide lines in Section 3.3.1. He will interactively be shown the artifacts that have to be included to provide a subsystem specification that is complete as a valid stand-alone and artifacts that might be of further interest will be added as reference. This corresponds to the method steps explained in Section 3.3.2.

The same kind of interactive dialogue will guide the user through the integration of an existing subsystem into a new surrounding system according to the method steps that have been described in Section 3.3.3. The
implementation of the AutoFOCUS3 is based on a common data model that is currently being developed.

6 Related Work

There are different categories of related work: One treats the decomposition of systems in software architecture, one issues the development of embedded systems with emphasis on the automotive domain, one treats the benefits and problems of reuse in general, one is concerned with product line development, one focuses on the reuse of specific types of artifacts, one is concerned with embedded systems. Apart from the well-known general paradigms like information hiding \[27\] and hierarchical structure \[11\] there are a number of profound approaches to software design, for example the Architecture Based Design method from SEI \[4\] and the successor method Attribute-Driven Design \[40\].

AUTOSAR \[3\] is an initiative for defining an architecture especially for the automotive domain, where automobile manufacturers, suppliers and tool developers developed an open and standardized automotive software architecture. A specially tailored model-based approach to automotive system design is presented by \[16\] and for embedded systems in general \[5\] propose the Metropolis Integrated Electronic System Design Environment.

Krueger presents an extensive survey of different approaches to software reuse, compares them using a taxonomy, and discusses the concept of abstraction \[24\]. Grinter discusses lessons from software reuse with regard to global coordination \[15\].

For product lines, Kang et al. approach reuse via discovery and exploitation of commonality \[22\]. Gomaa tailors domain modeling for the development of configurable distributed applications \[13\]. Fraunhofer ISE’s KobrA \[2\] \[1\] is an instance of PuLSE \[6\] and works with reusable components. Philips Research also developed a component-oriented approach \[26\] and discuss the change from product lines to product populations \[39\] for supporting more diversity. SPLC recent years

Related work that focuses more concrete on the actual reuse of artifacts is usually limited to a certain type of artifacts or models. Prieto-Díaz uses faceted classification to implement a software reuse library \[31\], Rising identifies pattern to reuse expertise \[34\], and Robinson et al. discuss benefits of and objections to simulation model reuse \[35\].

7 Summary and Future Research

We have outlined a method for the reuse of subsystems. We have explained our understanding of systems development, requirements engineering and tracing to prepare the ground for the method description. The method is accomplished through defining the subsystems by analyzing the different relevant types of criteria, their extraction with the specific set of related artifacts, and their integration into new surrounding-to-be-developed systems. We have discussed the additional aspect that product lines bring to the method and introduced the tool AutoFOCUS and our future implementation plans.

The next and most important piece of work is a case study to validate the concepts and evaluate the method steps. The study is from the automotive embedded systems domain and we make use of the above presented REM framework and its artifacts to gather a demonstrative example of application of the process and its results. This will on the one hand give us feedback on how good the approach works and on the other hand enable us to illustrate it with examples. Both will provide us with feedback and therefore help to improve the method.

Apart from the deeper investigation of the method details and the tool implementation, further research will include the application of the methods to different domains. Additionally, the integration of the method with other requirements engineering frameworks will be evaluated.

Acknowledgement: We would like to thank Sabine Rittmann and Judith Hartmann for their critical and constructive remarks on earlier versions of this paper.

References


