TUM

TECHNISCHE UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK

AMDiRE - Artefact Model for Domain-independent RE

Daniel Méndez Fernández, Birgit Penzenstadler, Manfred Broy, Jonas Eckhardt, Henning Femmer

TUM-I1327

AMDiRE - Artefact Model for Domain-independent RE
– *A small Cheat Sheet* –

D. Méndez Fernández[1], B. Penzenstadler[2], M. Broy[1], J. Eckhardt[1], and H. Femmer[1]

[1] Technische Universität München,
{mendezfe|broy|eckharjo|femmer}@in.tum.de,
[2] University of California, Irvine
bpenzens@uci.edu

Abstract. The various influences on processes and application domains make requirements engineering (RE) inherently complex and difficult to implement. When it comes to define an RE reference model for a company-wide use, we basically have two options: we can establish an activity-based RE approach where we define a blueprint of the relevant RE methods and description techniques, or we can establish an artefact-based approach where we define a blueprint of the RE artefacts rather than a blueprint of the way of creating the artefacts. In the last six years, we have established several artefact-based RE approaches and empirically underpinned the advantages of applying those approaches in industry. Those approaches remain, however, complex as they encompass various modelling concepts, and, in particular, incorporate their particularities of the different application domains, such as the one of business information systems. For this reason, we consolidated the different approaches and established the AMDiRE approach, i.e. the artefact model for domain-independent requirements engineering. AMDiRE includes a detailed artefact model that captures the basic modelling concepts used to specify RE-relevant information, tool support, and a tailoring guideline that guides the creation of the artefacts.
To provide a quick overview of the basic concepts of AMDiRE and the underlying principles, this report serves as a *Cheat Sheet*. This cheat sheet shall facilitate the basic understanding about the fundamentals in artefact orientation and provide an overview of a flexible artefact-based RE approach ready to be applied in practical contexts without focussing on technical details.

1 Motivation and Background

Requirements Engineering (RE) constitutes an important success factor for software and systems development projects as precise requirements are critical determinants of quality [4]. Although the discipline is known to be crucial for the success of every project, we still observe for many years companies struggling with their RE process. We experience unclear roles and responsibilities and a defined process that is obligatory for all projects, whereas it is too often performed mindlessly or even faked [13], without awareness of the reasons why a process step should (or should not) be executed, and without awareness of how to structure and specify the results. We could observe the impact on the RE artefacts, which often remain not reproducible, not measurable, incomplete, and inconsistent with no clear terminology [10]. A major reason for this circumstance is that many things are not clear from the beginning of a project, which makes the discipline—in combination with the diversity of characteristics in an application domain and the plethora of tools and techniques available—inherently complex and volatile. This is additionally hardened by potentially large amounts of requirements [15], which, if taking the current state-of-the-practices, are insufficiently structured in spreadsheets or relational databases, e.g. DOORS. The effects can be often observed in moving targets, incomplete and inconsistent requirements, and, finally, failed projects [9].

When it comes to define a company-wide *RE reference model*, i.e. an RE model that guides the activities of the discipline at the various project environments of a company, process engineers thus need to keep in mind the demand of supporting flexibility to cope with the various influences in individual project environments and precision to guide the reproducible creation of resilient specification documents.

In recent years, we have established several reference models for requirements engineering reflecting the artefact-based paradigm where we define a blueprint of the results rather than dictating complex activities, tasks, and methods. Those approaches have been disseminated into everyday practice, e.g. at Capgemini TS, Siemens, Bosch, BMW, or Cassidian. The AMDiRE approach—an artefact model for domain-inde-pendent RE—emerged as the result of consolidating those available approaches and the lessons we learnt during development, evaluation, and dissemination. The approach, however, is very complex as it captures the basic concept of software process models (roles or milestones) as well as the basic content definitions of the artefacts (defined as detailed, complex data models).

Aim of the Report

In this report, we provide a *Cheat Sheet* for AMDiRE, i.e. a brief overview of the most relevant aspects of artefact orientation and of the AMDiRE approach without focussing too much on the technical details of the approach. This cheat sheet serves three main purposes, namely provide an overview of:
1. requirements engineering in general (section 2)
2. the basic concepts of artefact orientation (section 3)
3. the AMDiRE approach (section 4)

The primary addressees of this document are therefore students of RE-related courses, as well as practitioners that apply AMDiRE or related artefact-based RE approaches.

Previously Published Guideline. The report at hand updates and extends the previously published mini-guideline to requirements engineering [14] with the new, consolidated artefact-based RE approach.

2 Requirements Engineering

In general, *requirements engineering* (RE) aims at iteratively eliciting, analysing, specifying, and validating & verifying the various requirements of interdisciplinary stakeholders. Taking a simplified view on the basic activities of RE, we distinguish subsequent major activities:
1. Elicitation: find out about the information
2. Analysis: think about it and structure it
3. Specification & Documentation: write it down
4. Validation & Verification: check back with stakeholder (validation) and verify whether the system fulfils the requirements (verification)

Requirements management (RM) aims at managing the specification and the usage of requirements over their whole life cycle, including, inter alia, their archiving and baselining, their systematic modifications, or their tracing and impact analysis during change requests.

Every requirement should run through the engineering phases and provide the information necessary to support the management phases. Each engineering phase is realised in so-called *tasks* or *methods*, i.e. sequences of modelling steps applied with a particular notation to specify an artefact's content or to modify it. An *artefact*, in turn, is a deliverable of major interest that abstracts from contents of a specification document. It is used as input, output, or as an intermediate result of a process definition (see also [8]).

In the AMDiRE approach, the information to be captured in RE is pre-defined in an *artefact model*. This artefact model serves as a checklist for the different aspects to be considered and builds the backbone to structure the specific tasks to be carried out during the requirements engineering phases. These tasks, for example, "elicit goals", are supported by methods and techniques. The results from these activities are captured in so-called *content items*, i.e. a logical grouping of the artefacts' contents according to the information needs of a specific stakeholder. Before introducing one artefact-based requirements engineering approach, we discuss the basic concepts of the underlying paradigm.

3 Artefact Orientation

Over the years, we could observe two basic paradigms to have emerged for the establishment of an RE reference model: activity orientation and artefact orientation.

The basic idea of *activity orientation* is to define a situation-specific RE process by a set of small steps, i.e. methods to be performed in a particular order by a specific set of roles. Although the research area [3,2] fostered the discussion on adaptable processes, it still does not cope with the various challenges today's RE environments have to face. Since available activity-centric contributions to RE focus on methods and description techniques as well as the complex dependencies between the methods rather than on clear result structures, contents, and dependencies among the results, project participants remain unaware of how to create consistent artefacts in their projects independently of underlying processes [7]. This shortcoming leads to the artefact-based paradigm.

The basic idea of *artefact orientation* is to establish a blueprint of the created results, their contents, and their dependencies rather than dictating complex processes and methods. That is, we abstract from the way of creating the results by the use of particular methods and modelling notations and specify *what* has to be done rather than dictating *how* to do something. As such, an artefact model abstracts from complex RE processes while capturing the various (modelling) concepts of the application domain. It thus provide a basis for a process-agnostic flexible backbone for project execution in which the project team agrees on the content and the structure of the artefacts to be created until a specific point of time while leaving open the way of creating the results, yet having a clear notion of responsibilities for each of the artefacts to be created.

An *artefact* is a deliverable of major interest that abstracts from contents of a specification document. It is used as input, output, or as an intermediate result of a process definition, i.e. the planned way of creating and modifying a set of artefacts via the application of particular tasks or methods (providing structured approaches to combining different description techniques [12]) in a particular sequence. An artefact has a particular status and underlies version control.

For each artefact, we capture two views: A *structure* view and a *content* view. The structure view captures for each artefact type (e.g. "requirements specification") the content items to be considered (e.g. "use case model"). For each content item, we define the content view via the modelling concepts, e.g. the elements and (content) relations of a use case model and different description techniques that can be used to instantiate these concepts, such as an UML activity diagram. The structure model thus gives a simplified view on the content and is used to couple the contents to the elements necessary to define a process, e.g.

roles, methods, and milestones relevant for a use case model. The content model supports awareness of the details necessary to specify the content as it provides the information about the modelling concepts and the relations, e.g. scenarios, actions and actors, which we use to create a use case model. Using both views in an artefact model as backbone of a process, we support a flexible RE to create precise result structures at project level (see Fig. 1).

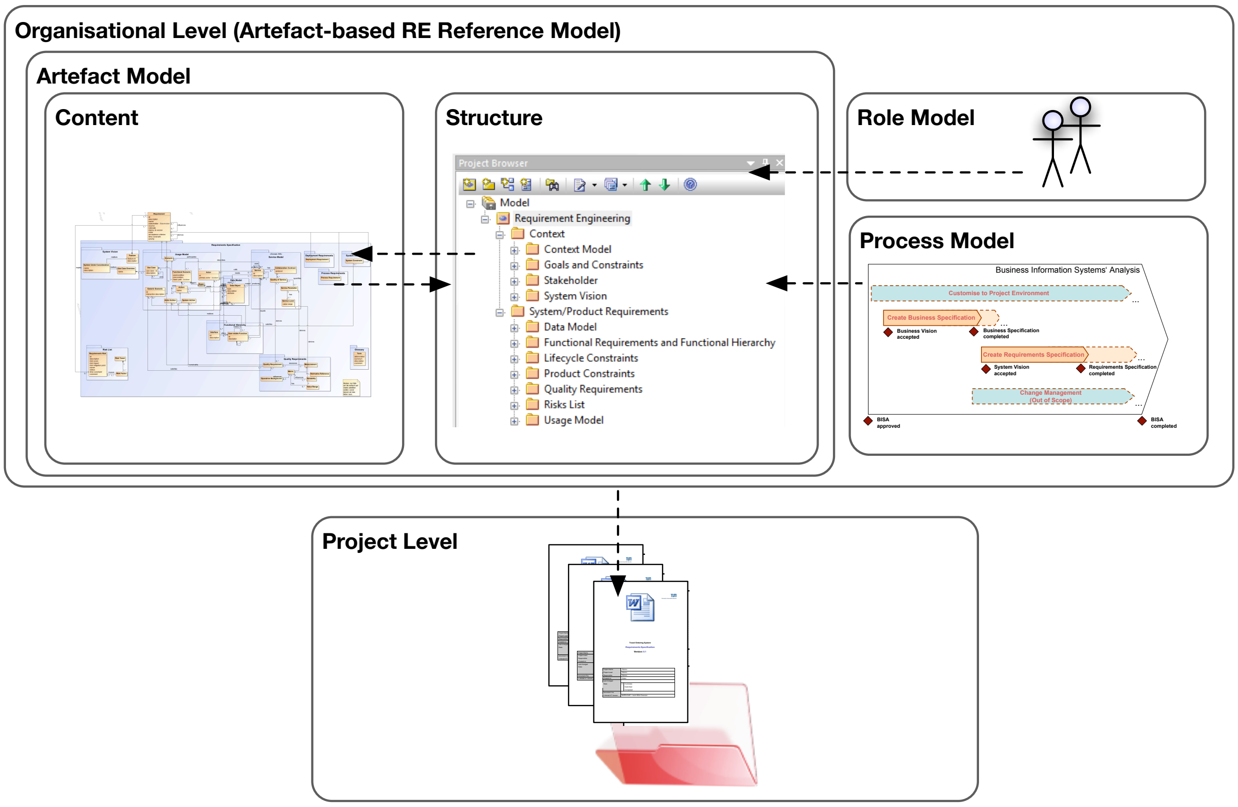
Organisational Level (Artefact-based RE Reference Model)
Artefact Model
Content
Structure
Project Browser
Model
Requirement Engineering
Context
Context Model
Goals and Constraints
Stakeholder
System Vision
System/Product Requirements
Data Model
Functional Requirements and Functional Hierarchy
Lifecycle Constraints
Product Constraints
Quality Requirements
Risks List
Usage Model
Role Model
Process Model
Business Information Systems' Analysis
Customise to Project Environment
Create Business Specification
Business Vision accepted
Business Specification completed
Create Requirements Specification
System Vision accepted
Requirements Specification completed
Change Management (Out of Scope)
BISA approved
BISA completed
Project Level

Fig. 1. Principles of artefact orientation.

Note that we consider (same as for activity-oriented approaches) a process necessary for artefact-based approaches, but instead of defining the process on basis of phases, activities and methods, we define the process on basis of the artefacts to be created. Even though the content model in the artefacts supports the precision of the results in the flexible process definition, the process itself remains undefined. Regarding the methods and description techniques for creating the contents (e.g. UML or natural text), we leave open which one to choose, as long as the contents and relationships proposed by the artefact model are specified.

In the following, we introduce one exemplary artefact-based approach: the AMDiRE approach. Please note, however, that its complete artefact model is too complex to explain within this cheat sheet. Therefore, we only ask you to remember the previous definition while the following descriptions provide an overview.

Further Reading. Further information on artefact orientation can be taken from the following literature:

1. Principles of artefact orientation: [8]
2. Empirical study on artefact orientation: [7]

4 AMDiRE Approach

AMDiRE emerged from a series of research co-operations and incorporates the basic concepts of artefact orientation as they have been introduced in the foregoing section. In the following, we first illustrate the basic components, before introducing the artefact model and its application.

4.1 Basic Components

Figure 2 shows the basic components of AMDiRE. The artefact model represents the back-

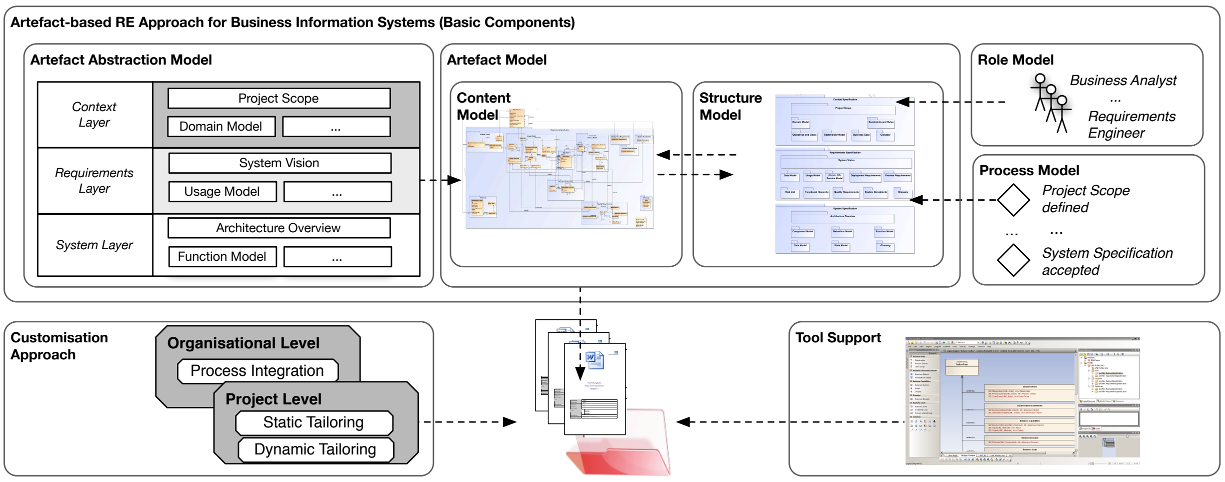

Fig. 2. Overview of the AMDiRE components.

bone of the approach and comprehends concepts used to specify the contents of the artefacts over three levels of abstraction (illustrated on the left side of the figure): *Context Layer*, *Requirements Layer*, and the *System Layer*. Each of those levels of abstraction features a specified number of content items that are detailed in concepts used for a step-wise refinement of the various (modelling) views we have on a system (see also [6]).

The context layer considers the context of a system, i.e. the domain in which to integrate the system such as the business domain with the business processes to be carried out. The requirements layer considers the system as a whole from a black-box perspective. That means, we specify the requirements on the system and the user-visible functionality from a perspective, in which the system is intended to be used, without giving details about its technical, internal realisation. The system layer provides the glass-box perspective on the internal (logical and technical) realisation of the system.

The artefact model is in the centre of our attention and consists of two basic models: the content model and the structure model. The content model abstracts from the modelling concepts used for a particular family of systems in a particular application domain over the defined levels of abstraction. The structure model gives a logical structuring to those concepts and is used for the integration with the role model and the process model (see also Sect. 3).

We distinguish in total three artefact types, each comprehending concepts to specify the desired information at one particular level of abstraction:

1. The *context specification* defines the context of the system under consideration including a specification of the overall project scope, the stakeholders, rules, goals, and constraints as well as a specification of the domain model. The latter comprehends, for example, business processes to be supported without defining how the system is intended to be used in context of those processes.
2. The *requirements specification* comprehends the requirements on the system under consideration, without taking a black-box view on the system. That means we specify requirements from a user's perspective without constraining the internal realisation of the system.
3. The *system specification* finally comprehends a glass-box view on the internal realisation of a system including a logical component architecture and a specification of the behaviour realisation with, e.g. functions and interfaces. While we consider the context and the requirements specification to address the problem space, the system specification addresses the solution space.

Figure 3 now shows the artefact types in relation to roles and responsibilities (left side) and in relation to milestones (right side).

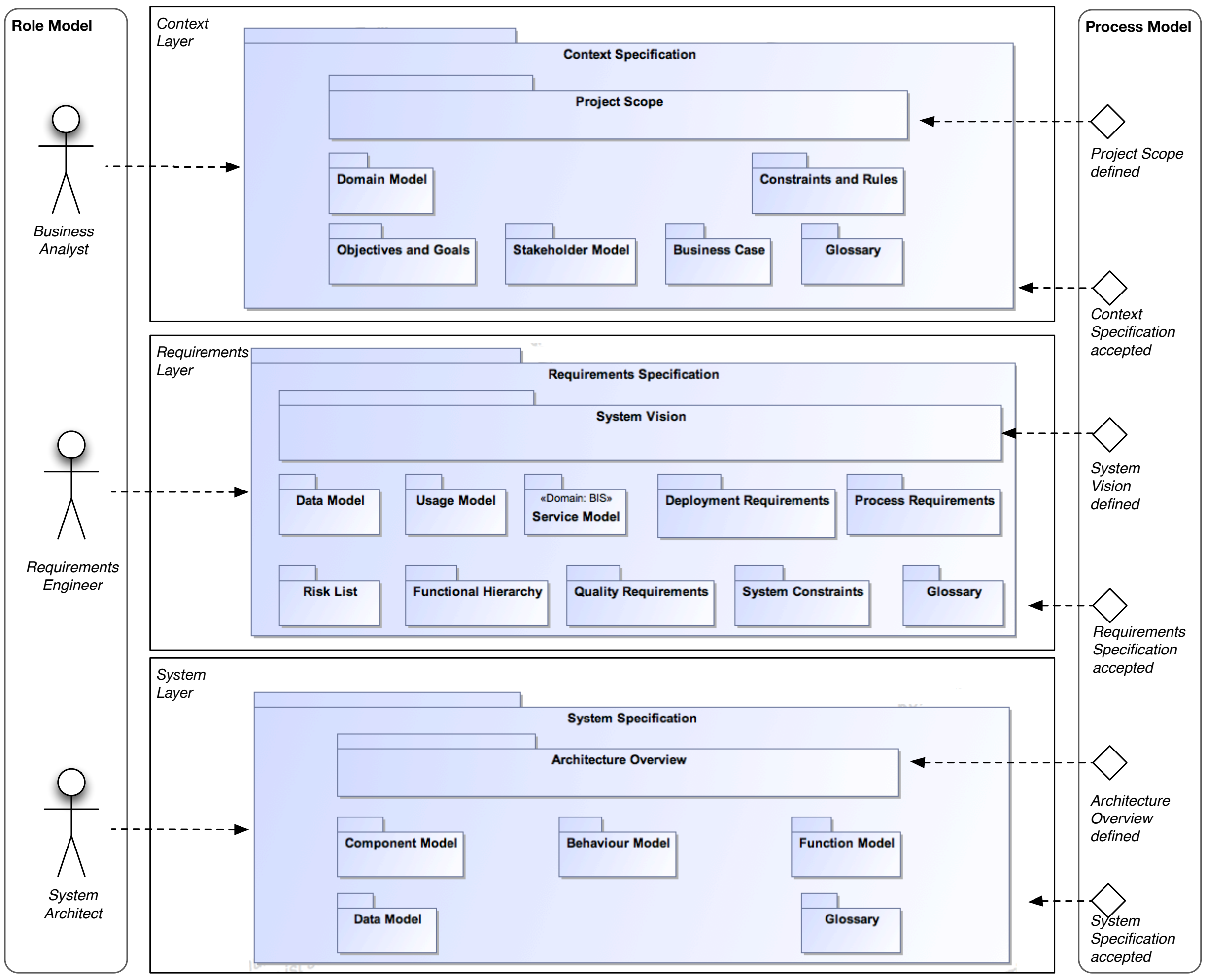

Fig. 3. Overview of artefacts types (structure model), roles, and milestones.

We use both the roles and the milestones to operationalise AMDiRE to a particular project setting. For each artefact type, we define one particular role, which has the responsibility for an artefact type, independent of other potentially supporting roles, and independent of whether same persons are assigned to different roles in a project.

1. The *Business Analyst* has the responsibility for the context specification and is expected to have the necessary domain knowledge, e.g. regarding the business processes, typical stakeholders as well as constraints and rules.
2. The *Requirements Engineer* has the responsibility for the requirements specification and serves also as a mediator between the business analyst and the system architect.
3. The *System Architect* has the responsibility for the system specification and is expected to have technical knowledge. In dependency to the application domain, we can further distinguish between a role for the logical architecture and a role for the technical architecture (e.g. in the area of business information systems).

For each artefact type, we furthermore define two milestones. The first milestones fixes the point in time in which the first content item is defined, thus, reflecting a certain maturity of the content in the artefact as the first content items serve the purpose of a summary for subsequent contents. For instance, the system vision in the requirements specification comprehends an overview of the major use cases; its definition and agreement indicate that the use cases are sufficiently defined to be further refined and modelled and, thus, allowing, for example, for first cost estimations based on function points. The second milestone of each artefact indicates the point in time when an artefact is finalised, respectively formally accepted.

Those milestones serve the purpose of a process integration and instantiation as they give us the opportunity to formally embed the artefacts into project-specific decisions to be taken at a specific point in time, such as when to conduct first cost estimations, when changes in the requirements should be formally defined via change requests, or when to take the contents in the specifications for a project classification and customisation (tailoring).

Tailoring AMDiRE (see lower part of Fig. 2) means to systematically decide under which situation to create which content item (or not) as the artefact model is no one-size-fits-all solution to every potential project setting. The tool support finally provides an UML profile-based extension of a modelling tool used to create the contents of the artefact model, e.g. concrete use cases and activity diagrams. Both the tailoring approach as well as the tool-support will not be discussed in detail as part of this cheat sheet.

4.2 AMDiRE Artefact Model

Figure 4 illustrates the artefact model of AMDiRE in a simplified manner.

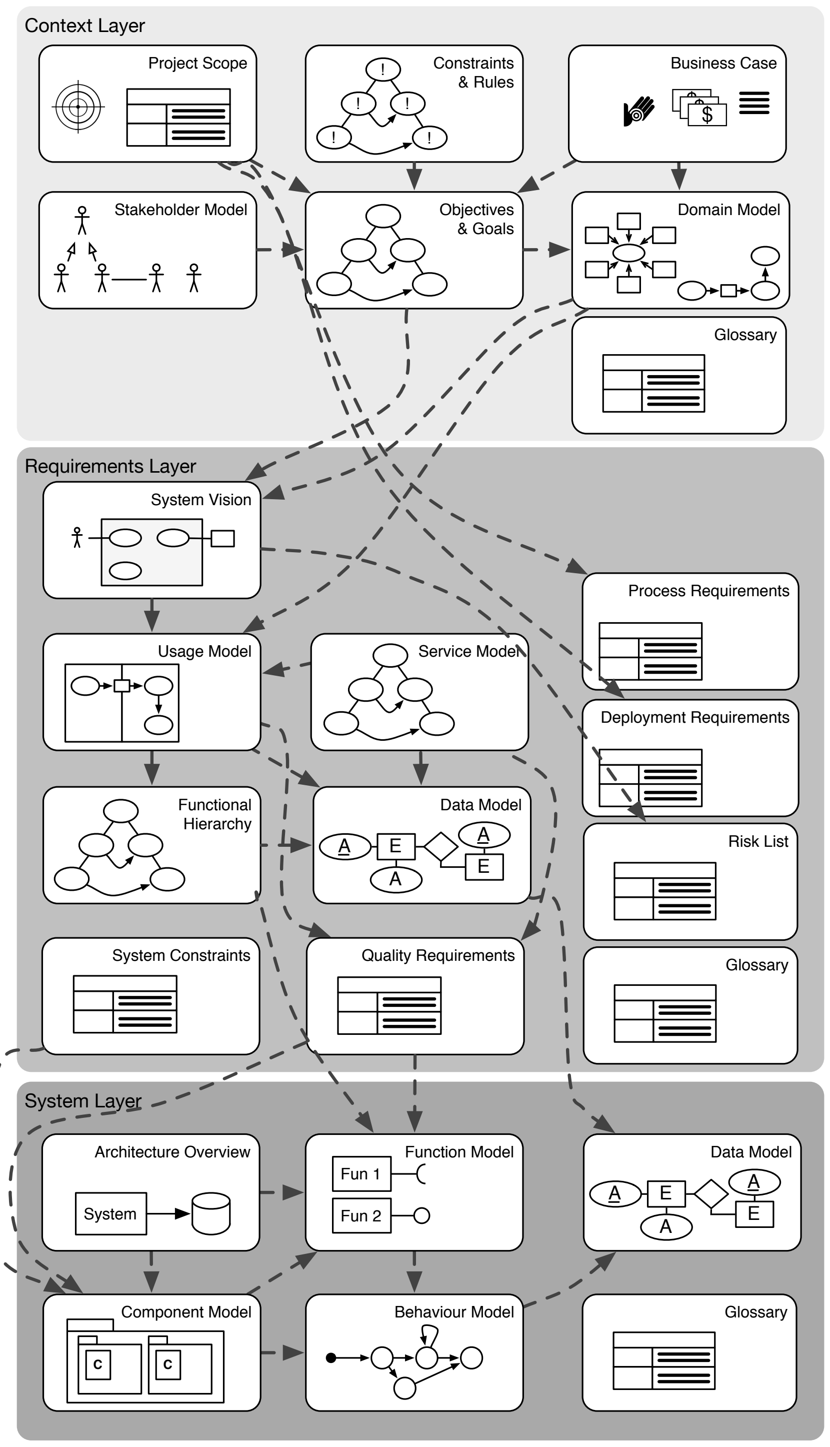

Fig. 4. Overview of the AMDiRE contents.

This illustrative overview already depicts some of the most important relations between the content items, e.g. from the *Objectives* to the *System Vision* or from the *Domain Model* to the *Usage Model*. At the same time, it is crucial to note that this cheat sheet can only give a brief overview, but not provide the content model in detail. We intentionally do not define the concept model in detail as we shall rather discuss the basic content items and their basic dependencies.

Similar as done with Fig. 3, we distinguish again three artefact types, which we organise according to the previously defined three levels of abstraction. Each artefact type comprehends a set of content items that capture the contents in the documents or data sets to be created during requirements engineering. All content items are briefly explained in the table in the appendix.

4.3 Applying AMDiRE

The process for applying AMDiRE strongly depends on the overall software process into which we integrate the approach, the organisational culture and, in particular, the project setting. We thus cannot define a particular process to apply AMDiRE that, as a matter of fact, is one major reasons why we follow the artefact-based philosophy.

However, when creating the artefacts as they are defined in the AMDiRE artefact model, we need to follow the content-related dependencies in the content model that define, to some extent, causal relationships between the contents' creation; for instance, to define a use case model based on business process models, we first need to specify business processes (to some extent), or participating user groups. In the following, we thus give a simplified and idealised (green field) engineering view on the creation of the content items following the relations depicted in Fig. 4.

Starting Point The most common starting point for requirements engineering is a customer who issues the desire for a specific software solution. In the content model, this is one of the stakeholders. They have an idea of a project, define the scope of the project and afterwards build a system vision. You get a first version of the system vision from your customer and you get more stakeholders to talk to.

Finding the Stakeholders One major pitfall for requirements engineering is to have an incomplete list of stakeholders consequently resulting in incomplete requirements or constraints. To overcome this problem, we need to systematically gather the people who have a potential interest in the system and defining a concrete stakeholder model, also to provide a key reporting line in RE. For every stakeholder, choosing the right vocabulary to reach out to them is important. Whether or not a stakeholder has technical knowledge, is involved with marketing, or represents legal concerns, we must take their requirements into account. Important common types of stakeholders that should be checked for every system are:
- Customer
- User groups / actors
- Marketing and sales
- Workers' council
- Legislation
- Developers (hardware and software)
- Technical cupport and maintenance

Further Reading. A helpful reference for further reading is the taxonomy of stakeholders by Ian Alexander, called onion model [1].

Agreeing on a Scope All stakeholders should agree on a common project scope and system vision. This implies that the system vision has to be understood by every stakeholder involved—including non-technicians. We advise to use a so-called rich picture [11] as illustration. A rich picture captures the key elements of the system vision in (self- explanatory or labelled) icons and depicts their interrelation and is especially useful as basis for discussion in workshops and meetings. In documentation, i.e. after the stakeholders have agreed on one version of the rich picture, it should be accompanied by an explaining paragraph in natural language.

Specifying Goals, Usage Model, and Constraints From the stakeholders, we gather the goals. Goals can be business goals, usage goals, or system-related goals such as "increase the maintainability". The two most common forms of documenting goals are natural language text and goal graphs [16]. Apart from the stakeholders, there are other sources of information, usually documents. Some important information sources are:

- Legacy systems and user documentation
- Laws, standards, and regulations
- Customer complaints

Having documented the goals, we use them to specify the domain model of the system with a particular focus on business processes, which define the work to be carried out. While doing so, we get first impressions how a future system is intended to be used in context of those processes. In case of embedded reactive systems, we can directly use the goals to motivate the usage scenarios in our usage model or functional hierarchy. To this end, we use the goals that refer to the services to be provided by the system and refine them into descriptions of usage behaviour. This often requires additional input from both stakeholders and other information sources.

The results of this analysis are captured in the content item "usage model". The intended usage of the system can be captured either in service descriptions or in use cases and scenarios. The usage model is a black box specification of the behaviour that hides any realisation detail of the system, i.e. the logical component architecture and internal functions. Constraints describe restrictions that arise from the business context (like management or laws) or from the system's operational environment (like hardware constraints). Both types have to be listed with references to their original source.

Finally, we use this information to draw a first component architecture, i.e. components, ports and interfaces, and the identified (user-visible) services the system shall offer.

Quality Assurance and Acceptance With quality assurance, we refer to the validation of the RE artefacts. With the artefact model, we ensure, at least to some extent, a constructive quality assurance as we ensure that the requirements are documented in a certain syntactic quality. The analytical quality assurance is usually done within a quality gate by a person not directly involved in the RE activities. We check for quality attributes like understandability or precision. The formal acceptance finally involves the acceptance of the artefacts by the customer.

Requirements Management Your requirements are likely to be changed by the customer and other stakeholders during development. Therefore, it is important to establish a proper change management process that keeps your requirements specification consistent and their change history traceable. Best practice is issuing change requests and deciding on their relevance. These decisions, as well as other decisions taken during development, should be documented for future reference. The usage of the artefact model and, in particular, the content-related dependencies finally support traceability, which support impact analysis and (syntactic) consistency among the artefacts.

References

1. Alexander, I. A Taxonomy of Stakeholders: Human Roles in System Development. *International Journal of Technology and Human interaction*, 1(1):23–59, 2005.
2. C. Braun, F. Wortmann, M. Hafner, and R. Winter. Method Construction - A Core Approach to Organizational Engineering. In *Proceedings of the 20th ACM symposium on Applied computing (SAC '05)*, pages 1295–1299. ACM New York, NY, USA, 2005.
3. S. Brinkkemper. Method Engineering: Engineering of Information Systems Development Methods and Tools. *Information and Software Technology*, 38(4):275–280, 1996.
4. M. Broy. Requirements Engineering as a Key to Holistic Software Quality. In A. Levi, E. Savas, H. Yenigun, S. Balcisoy, and Y. Saygin, editors, *Proceedings of the 21th International Symposium on Computer and Information Sciences (ISCIS 2006)*, volume 4263, pages 24–34. Springer-Verlag Berlin, 2006.
5. M. Broy, M. Feilkas, J. Grünbauer, A. Gruler, A. Harhurin, J. Hartmann, B. Penzenstadler, B. Schätz, and D. Wild. Umfassendes Architekturmodell für das Engineering eingebetteter Software-intensiver Systeme. Technical Report TUM-I0816, Technische Universität München, jun 2008.
6. T. Gorschek and C. Wohlin. Requirements Abstraction Model. *Requirements Engineering*, 11(1):79–101, 2006.
7. D. Mendez Fernandez, K. Lochmann, B. Penzenstadler, and S. Wagner. A Case Study on the Application of an Artefact-Based Requirements Engineering Approach. In *Proceedings of the 15th International Conference on Evaluation and Assessment in Software Engineering (EASE 2011)*, pages 104–113. Institution of Engineering and Technology (IET), 2011.
8. D. Mendez Fernandez, B. Penzenstadler, M. Kuhrmann, and M. Broy. A Meta Model for Artefact-Orientation: Fundamentals and Lessons Learned in Requirements Engineering. In D. Petriu, N. Rouquette, and O. Haugen, editors, *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems (Models)*, volume 6395, pages 183–197. Springer-Verlag Berlin Heidelberg, 2010.
9. D. Mendez Fernandez and S. Wagner. Naming the Pain in Requirements Engineering: Design of a global Family of Surveys and first Results from Germany. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE 2011)*. ACM Press, 2013.
10. D. Mendez Fernandez, S. Wagner, K. Lochmann, A. Baumann, and H. de Carne. Field Study on Requirements Engineering: Investigation of Artefacts, Project Parameters, and Execution Strategies. *Information and Software Technology*, 54(2):162–178, 2012.
11. Monk, A. and Howard, S. The Rich Picture: A Tool for Reasoning about Work Context. *Interactions*, 5(2):21–30, 1998.
12. B. Nuseibeh and S. Easterbrook. Requirements Engineering: A Roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 35–46, New York, NY, USA, 2000. ACM.
13. D. L. Parnas and P. C. Clements. A Rational Design Process: How and Why to fake it. *IEEE Transactions on Software Engineering*, 12(2):251–257, 1986.
14. B. Penzenstadler. Mini-Guideline to Requirements Engineering. Technical Report TUM-I126, Technische Universität München, 2012.
15. B. Regnell, R. B. Svensson, and K. Wnuk. Can We Beat the Complexity of Very Large-Scale Requirements Engineering? In *Proceedings of the 14th international conference on Requirements Engineering: Foundation for Software Quality*, REFSQ '08, pages 123–128. Springer-Verlag, 2008.
16. A. van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications.* Number ISBN-13: 978-0470012703. Wiley & Sons, 2009.

Content Item	**Exempl. Notation**	**Description**
Project Scope	Natural text	Problem description and a statement of intent, i.e. a conclusion of the objectives of a potentially resulting project.
Constraints & Rules	Natural text, graphs	Restrictions that can influence each other either in form of support or a conflict. We distinguish constraints as not negotiable restrictions and rules as conditional standard procedures.
Stakeholder Model	UML actor hierarchy, tables	Stakeholders comprehend individuals, groups, or institutions having the responsibility for requirements and a major interest in the project. User groups are a specialisation of stakeholders interacting with the system.
Business Case	Natural text	Described and detailed using cost, value, and risk. The business case satisfies the statement of intent from the project scope and rationalises the goals in the content item objectives and goals
Objectives & Goals	Goal graphs (e.g. KAOS)	Each goal, whether it is a business goal, a usage goal, or a system goal, is issued by a stakeholder. Goals satisfy the statement of intent [16], they build a hierarchy, and they can influence each other in terms of conflicts, constraints, or support.
Domain Model	UML activity diagrams or BMPN	The external systems, that interact with the system under development, compose the domain model. For business information systems, the domain model is extended with a business process model represented in various types of activities that need and produce business objects. A business process model is a collection of all instances of the activities and their (causal) relations.
Glossary	Structured text	This content item contains all important terms for the system under consideration, their abbreviation, synonyms, and description. It shows up as well in requirements specification and the system specification as more terms are added over the course of the project.
System Vision	Rich picture	The system vision comprehends the system context of the system under consideration, which is intended to realise a number of features. A feature is a prominent or distinctive user-recognisable aspect, quality, or characteristic of a system. In addition, we specify an use case overview, i.e. a (potentially) graphical overview of the use cases.
Usage Model	Structured text, UML activity diagrams	This content item details the use case overview of the system vision in its use cases. We distinguish services and use cases. Both concepts are means to describe (black box) system behaviour. Use cases describe sequences of interaction between actors (realising user groups) and the system as a whole.
Service Model	Graphs	This content item is relevant for the domain of business information systems and specifies services. Services describe a logical representation of a use case, not necessarily involving actors or concrete sequences of interaction.
Data Model	UML class diagrams	The data model contains all objects processed as part of functions and interaction scenarios.
Functional Hierarchy	Graphs, I/O-tables	The user-visible functions complement the services from the service model and realise the system actions from the usage model. Functions are organised in a hierarchy to build the transition to the system specification as they describe a logical representation of a system action and offered by typed interfaces.
Quality Requirements	Natural text	Quality requirements are assessed by measurements that can be either a normative reference (e.g. style guide) or a metric. Quality requirements constrain system properties and can be formulated with (more abstract) generic scenarios.
Deployment Requirements	Natural text	Deployment requirements describe demands towards the deployment procedure, constraining the process design of the deployment, and the technical environment during initial launch of the system or specific parts of it.
System Constraints	Natural text	The system constraints describe logical and technical restrictions on a system's architecture and its quality by means of assessable system quality requirements. Hence, we see a system as a grey box rather than as a glass box, since we restrict systems' internals, but do not consider their logical structure.
Process Requirements	Natural text	Process requirements constrain the content and / or structure of selected artefact types and the process model, i.e., the definition of the milestones regarding time schedules, used infrastructure, and standards.

Risk List	Natural Text	The Risk list includes a description of all risks that are related to project-specific requirements. Each risk is caused by a risk factor.
Architecture Overview	Component diagram, table	The architecture overview includes the component overview as well as the major functions that summarise most important functions of the system.
Function Model	Graphs, tables	The system functions offered by the system interface realise the user-visible functions from the functional hierarchy and provide a big picture of the functions offered by the components.
Component Model	Component diagram	The component model describes the logical component architecture in detail including ports (building typed interfaces). Components can be decomposed into more further sub-components [5].
Behaviour Model	Automata	The behaviour of components is specified with by events, states and state transitions. The resulting state machines specify the behaviour and the used data elements of the data model.
Data Model	UML class diagram, dictionary	We specify data elements of a certain type and their relations as they result from the behaviour model.