Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

Irina Gogoryan
MASTER THESIS

Meta Model to support requirements specification creation

Submitted on 30.08.2022

Supervisor: Julia Krause, M. Sc; Prof. Dr. Dirk Riehle, M.B.A.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander University Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

_____

Nürnberg, 30.08.2022

License

_____

Nürnberg, 30.08.2022

Abstract

Requirements elicitation is an initial phase and an essential cornerstone in a software development lifecycle. The requirements and, in particular, the software requirements specification form a base for the design and development of software.

Several surveys and studies have identified a range of problems, whereby one of them showed that incomplete and hidden requirements are among the most common issues that can cause financial problems and be a reason for a software development project's failure. (Fernández et al., 2017) Therefore, software requirements and specification quality play an essential role in a successful software development project.

In this thesis, we examine the possibility of creating a natural language requirements specification using a qualitative data analysis method – QDAcity RE – which makes use of a metamodel. This method has already been applied and tested for domain modelling. We expect that this method will further contribute to the efficient creation of a software requirements specification to reduce the failure in later phases of software development.

Contents

Abstract 3
Contents 4
List of Figures 5
List of Tables 6
1 Introduction 7
1.1 Original thesis goals 7
1.2 Changes to thesis goals 7
1.3 Structure of the thesis 7
2 Related work 8
2.1 Definition of relevant terms 8
2.1.1 Grounded theory 8
2.1.2 Software requirements 8
2.2 Creation of requirements specification 11
2.2.1 Using QDA-based methods in software engineering 11
2.2.2 Using metamodels to create a requirement specification 11
2.3 The QDAcity-RE method 13
2.4 Preliminary work 14
3 Research design 16
3.1 Research question 17
4 Used data 18
5 Research approach and results 19
5.1 Analysis of the interviews 19
5.2 Comparison of code systems 20
5.2.1 Quantitative comparison 20
5.2.2 Qualitative comparison 23
5.3 Creation of specification 25
5.3.1 Challenges in creating a specification based on the code system for domain modelling 29
5.4 Proposal for a unified metamodel 30
6 Discussion 32
7 Limitations 34
8 Conclusions 35
Appendix A - The code system for domain modelling 36
Appendix B - The code system for requirements specification 40
Appendix C - Requirements Specification 45
References 60

List of Figures

Figure 1: Prototype SRS outline (IEEE Computer Society, 1998) 10
Figure 2: Metamodel for Web Requirements Engineering (WebRE), (Escalona & Koch, 2007) 12
Figure 3: Metamodel for NFRs, FRs, and their relations, (Kassab et al., 2009) 12
Figure 4: Agile RE metamodel (Schön et al., 2019) 13
Figure 5: The QDAcity-RE Process for Structural Domain Modeling (Kaufmann & Riehle, 2019) 14
Figure 6: Metamodel, (Salow, 2016) 15
Figure 7: SOPHIST template - English version (Kluge & SOPHIST GmbH, 2016) 16
Figure 8: SOPHIST template - German version (SOPHIST) 16
Figure 9: Project statistics from QDAcity 20
Figure 10: Project information from MAXQDA 20
Figure 11: List of the documents in MAXQDA 21
Figure 12: Comparison of codings by the number of coded text segments 21
Figure 13: Colours of codes in the code system for the domain model in MAXQDA 22
Figure 14: Colours of codes in the code system for the for requirements specification in QDAcity 22
Figure 15: Comparison of code systems by the number of codes in different hierarchical levels 23
Figure 16: FunctionalMASTER (Kluge & SOPHIST GmbH, 2016) 26
Figure 17: Requirement 3.6 - Screenshot from the requirements specification in QDAcity 26
Figure 18: Requirement 2 - Screenshot from the requirements specification in QDAcity 26
Figure 19: Requirement 3.11 - Screenshot from the requirements specification in QDAcity 27
Figure 20: FunktionsMASTER with conditions (Kluge & SOPHIST GmbH, 2016) 28
Figure 21: Completeness analysis of the requirements specification 28
Figure 22: Unified metamodel 30

List of Tables

Table 1: Code System Meta Model Elements transferred to Domain Meta Model Elements (Kunz, 2015) 15
Table 2: Metamodel elements and corresponding colours 19
Table 3: Steps for creating a sentence based on the SOPHIST template 25
Table 4: Adjusted table for metamodel elements and corresponding colours 31

1 Introduction

1.1 Original thesis goals

Qualitative Data Analysis (QDA) is a method of analysis used in qualitative research and focuses on extracting relevant information from qualitative data, such as interviews with stakeholders. The Professorship for Open Source Software at the Friedrich-Alexander-University at Erlangen Nuremberg developed QDAcity, a cloud-based QDA software which researchers use to perform qualitative data analysis. Furthermore, in previous research within the research group, QDA has been examined as a new method for domain analysis, called the QDAcity-RE method. The result of the QDAcity-RE method is a conceptual domain model created based on a metamodel.

This thesis aims at analysing and adapting the existing metamodel to support the creation of a natural language requirements specification. The refinements should lead to a metamodel that can be used to create a conceptual model and a natural language requirements specification.

1.2 Changes to thesis goals

The initial goals of this thesis were not changed. However, the approach to reaching those goals was altered during the writing process.

1.3 Structure of the thesis

Chapter 1 of this thesis presents an introduction to the topic showing the goals and the structure of the thesis. The remainder of this master thesis is structured as follows: chapter 2 presents the related work and defines relevant terms, including grounded theory, requirements, and software requirements specification. Moreover, in chapter 2, the QDAcity-RE method and a preliminary master thesis are presented. Chapter 3 overviews our research design and defines the research questions. The data used in this thesis is presented in chapter 4. The following chapter, 5, shows the research approach and the results. It includes the subchapters for analysis of the stakeholder interviews, quantitative and qualitative comparisons of code systems, describes the creation of the requirements specification, and proposes a unified model for creating a domain model and a requirements specification. Finally, chapters 6, 7 and 8 present the discussion part, limitations of the thesis and conclusions accordingly.

2 Related work

2.1 Definition of relevant terms

2.1.1 Grounded theory

Grounded theory is a "qualitative research method that uses a systematic set of procedures to develop an inductively derived grounded theory about a phenomenon" (Corbin & Strauss, 1990a). This theory was discovered by Glaser and Strauss (1967) and was further developed by different researchers.

Coding is a critical process in grounded theory. In their article, Corbin and Strauss present the following three stages of coding and describe them as follows:

1. Open coding
2. Axial coding
3. Selective coding.

During open coding, incidents indicating a phenomenon are identified by analysing the underlying data. The data units are coded with a code which is called concept. Consequently, events, actions and interpretations are conceptually labelled, and if similarities are detected among the concepts due to constant comparison, then they are grouped in categories. These codes are not hierarchically structured in this phase and are not connected to each other analytically.

In the second phase - axial coding – similar categories are grouped into a more abstract category. In this stage, conditions, context, strategies and consequences are taken into account. In other words, axial coding is about interconnecting the categories or codes identified during the open coding stage.

Finally, one or a couple of main categories are selected in the selective coding phase to represent the theory's central analytic idea.

Several key elements and procedures in grounded theory are used in qualitative data analysis. Corbin and Strauss defined eleven canons and procedures for grounded theory. For example, the canon "concepts are the basic units of analysis", and the procedure of memo writing to store important notes concerning the codes are also applicable in QDA.

2.1.2 Software requirements

There are different types of models available in the software development lifecycle process. Each of them consists of a set of phases and activities. However, all of the models have similarities and usually comprise the following main activities: planning and visualization, requirement analysis, software modelling and design, coding, documentation, testing, as well as deployment and maintenance of software. Regardless of whether the waterfall model, iterative model, V-shaped model, spiral model or agile model is applied during software development, the software development lifecycle phase dealing with requirements is very initial. Usually, requirements analysis is the very first phase, among others. Therefore, the quality of requirements analysis is of great importance, as the requirements are the base for the design and development of software. (Rastogi, 2015)

Several definitions are available for software requirements, which form the core of requirements analysis. One of them defines a software requirement as follows: "At its most basic, a software requirement is a property that must be exhibited by something in order to solve some problem in the real world. It may aim to automate part of a task for someone to support

the business processes of an organization, to correct shortcomings of existing software, or to control a device—to name just a few of the many problems for which software solutions are possible." (Bourque & Fairley, 2014). Compared to the latest, SOPHIST (2016) proposes a broader definition: "A requirement is a statement concerning a property or the performance of a product, a process or the people involved in the process".

A range of sources forms a base for requirements elicitation. The principal sources are the goals and high-level objectives of the software, domain knowledge, business rules, the operational and organizational environments, and stakeholders, such as users and customers, who impact the desired software. Gathering the requirements is also executed with the use of various techniques. The main ones are stakeholder interviews, scenarios, prototypes, facilitated meetings, observations, user stories and other techniques such as analysing competitors' software products. (Bourque & Fairley, 2014)

Requirements can be categorized based on different aspects. This thesis mainly uses the requirements categories based on the software functionality. Based on the functionality, there are two types of requirements: functional and non-functional. (Chung & do Prado Leite, 2009) Functional requirements describe the functionality of a system or system services, and they depend on the type of software and expected users. (Kluge & SOPHIST GmbH, 2021) On the contrary, the non-functional requirements define the system's constraints and quality attributes. (Bourque & Fairley, 2014) They are further classified into the following categories: user interface requirements, technological constraints, quality requirements, legal and contractual requirements, requirements for activities and services, and requirements for other deliverables. (Kluge et al., 2016)

Compared to functional requirements, the non-functional ones are often overlooked and missed because system development primarily deals with the question of what a system does and which functionality it has, not how it behaves. However, non-functional requirements are, in some cases, more critical than functional requirements because if the non-functional requirements are not fulfilled as initially required, the developed system can be unsatisfying and even useless. (Franch, 1998; Pozgaj et al., 2003)

A requirement can be documented in different ways. Model-based, natural language and also mixed types of requirements documentation are possible. Apart from classifying the requirements based on software functionality, three perspectives on requirements can be differentiated: structural, functional and behavioural perspectives. In particular, model-based requirements documentation gives the opportunity to inspect the expected software from different perspectives. For this purpose, several types of UML diagrams are used. (Kluge et al., 2016)

The abbreviation UML stands for the Unified Modelling Language. With the help of UML, IT professionals are designing a software project. It is a graphical language for specifying, visualizing, constructing and documenting the artifacts of a software system. (Shen et al., 2002) UML diagram represents two different views of a system model: static model or a structural view and dynamic or behavioural view. (Bell, 2003) There is a range of different UML diagrams available. Among them are the use case diagram, the class diagram, the activity diagrams and the statechart diagram. The use case and the class diagrams are also used in visualizing requirements. (Bell, 2003; Kluge et al., 2016)

A class diagram shows a set of classes and their relationships. A class is depicted as a rectangle in a class diagram, whereby attributes and operations can be assigned. (Bell, 2003) Depending on the type of relationships between classes, association, aggregation, generalization and dependencies can exist. (Genero et al., 2005)

In use case diagrams, the main constituents are the actors, use cases and the relationships between actors and use cases, as well as the relationships between different use cases. (Bell,

2003) This requirements specification includes a set of use cases describing the users' interactions with the software. In this thesis, we will not concentrate on the use case-based approach in the research part.

Overall, the process of requirements engineering is defined as “a structured set of activities which are allowed to derive, validate and maintain a systems requirements document.” (Kotonya, G., Sommerville, Ian, 1998) That means that the central core of this phase is the requirements specification document.

When starting a software development project, it is essential to complete the requirements documentation. A requirement specification for a software system is a documented description of the behaviour of a desired system. It means it specifies what the system should and should not be able to do. The requirements defined in the software requirements specification reflex the needs of a customer for a system that help to solve some problems. The specification serves as a base for agreement between customers and contractors or suppliers. (Bourque & Fairley, 2014)

The IEEE Computer Society mentioned in the standard for recommended practice for software requirements specifications a set of characteristics which are needed to form a good software requirements specification. According to that set, the specification should be correct, unambiguous, complete, consistent, ranked for importance and/or stability, verifiable, modifiable and traceable. (IEEE Computer Society, 1998)

There are various options for a requirements specification structure available in the literature. However, according to IEEE, a good specification should at least consist of the information in the sections presented in Figure 1.

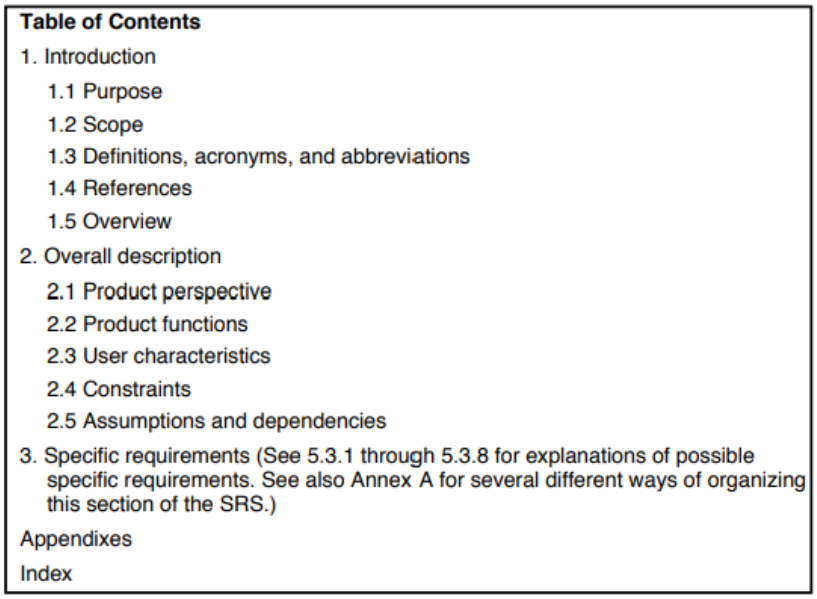


Figure 1: Prototype SRS outline (IEEE Computer Society, 1998)

This thesis focuses on creating the parts of a specification which include the functional and non-functional requirements. Therefore, not all of the essential parts of a good software requirements specification are covered and discussed in this thesis.

2.2 Creation of requirements specification

2.2.1 Using QDA-based methods in software engineering

Several authors have analysed the applicability of the grounded theory in different software engineering processes, including in the requirements engineering phase.

For example, Carvalho et al. (2005) investigated the application of the grounded theory method in the software engineering process research domain by comparing two process models. One was introduced by a software engineer, and the other by a qualitative data analyst. Although they found significant differences between these two models, applying the methods such as constant comparison can lead to an improved modelling process.

Chakraborty and Dehlinger (2009) used the grounded theory method to determine enterprise system requirements in their work. They derive a UML diagram from a description of a university support system. In the class diagram, the features and information about the implementation represent classes, while the relationships between classes are not mentioned. The authors pointed out in their work that further research should examine how the grounded theory method can make the modelling and representation of initial enterprise requirements easier and help identify gaps in requirements engineering.

The application of the grounded theory was also tested in the studies of Halaweh (2012). According to them, categories and their relationships can be opposed to classes and their relationships in class diagrams. Therefore, the informal model, produced based on grounded theory, can be turned into a formal model, such as a UML class diagram. Halaweh stated that the grounded theory method could contribute to better communication between the analyst and the user and between analysts and developers. It might also be used for better understanding the users' needs and nontechnical aspects related to information systems development.

In another study, Chakraborty, Rosenkranz, and Dehlinger (2015) developed a grounded and linguistic-based requirements analysis procedure (GLAP) for a systematic and traceable elicitation of non-functional requirements in order to support sensemaking in requirements engineering.

2.2.2 Using metamodels to create a requirement specification

Using different models for creation of requirements specification is common in research and practice. Several researchers have developed and used metamodels specifically in requirements analysis. A metamodel for the usage in information systems can be defined as follows: "In the context of information systems, a metamodel contains statements about the constructs used in models about information systems. The statements in a metamodel can define the constructs or can express true and desired properties of the constructs. Like models are abstractions of some reality, metamodels are abstractions of models." (Jeusfeld, 2009)

For example, Escalona and Koch (2007) suggested a metamodel specifically for the requirements of web systems. It is introduced in Figure 2. The UML metamodel consists of two packages: behaviour and structure. It provides key concepts to produce web requirements specifications, which are represented as UML meta classes.

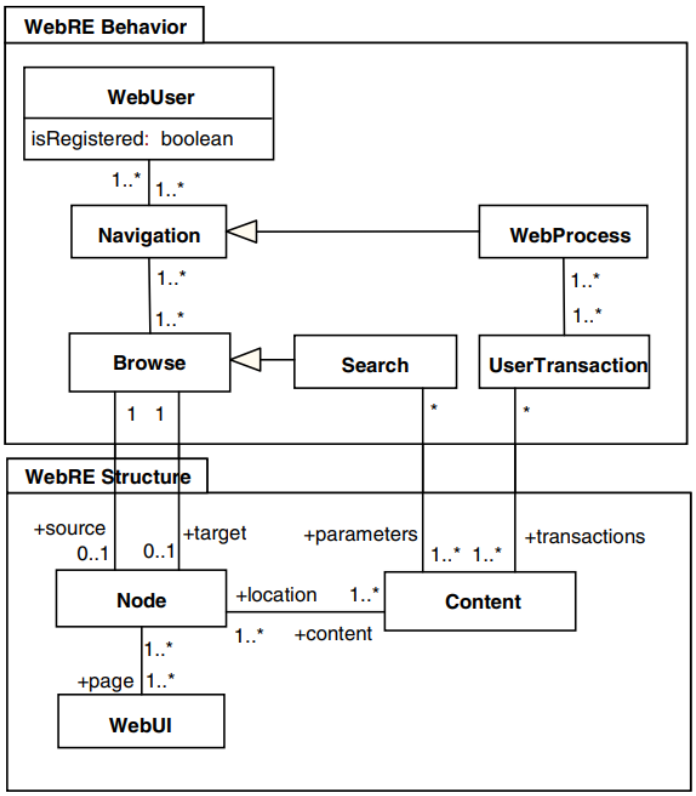


Figure 2: Metamodel for Web Requirements Engineering (WebRE), (Escalona & Koch, 2007)

Another study proposed a metamodel that explicitly focuses on tracing non-functional requirements and helps reveal the relations between functional and non-functional requirements and within those two categories of requirements. However, it was not validated by empirical research. (Kassab et al., 2009) This metamodel is shown in Figure 3.

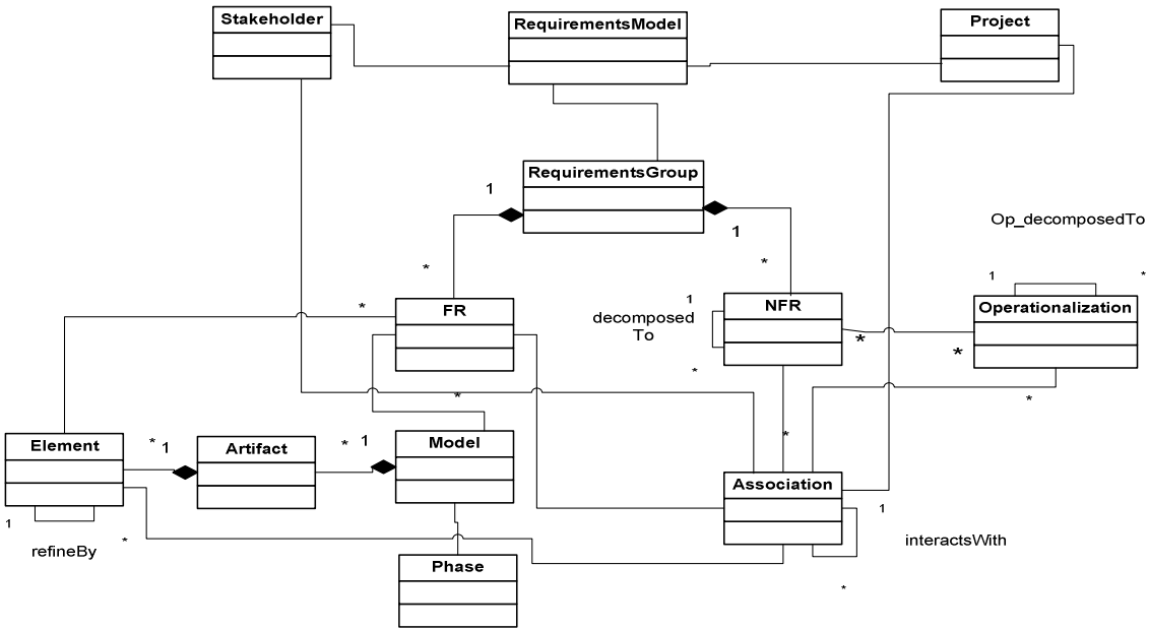


Figure 3: Metamodel for NFRs, FRs, and their relations, (Kassab et al., 2009)

Among others, there is a metamodel, that represents requirements patterns with variability modelling and traceability of software artifacts. It is developed in the research of Ya'u et al. (2016). Improving the reuse of requirement patterns while integrating concepts of software product line and model-driven engineering is the primary purpose of this metamodel.

In another research on metamodeling, the authors proposed a metamodel for agile requirements engineering, introduced in Figure 4. This model can be implied for modelling new process models in the field of agile software development. Moreover, this metamodel can help evaluate and improve the existing processes. (Schön et al., 2019)

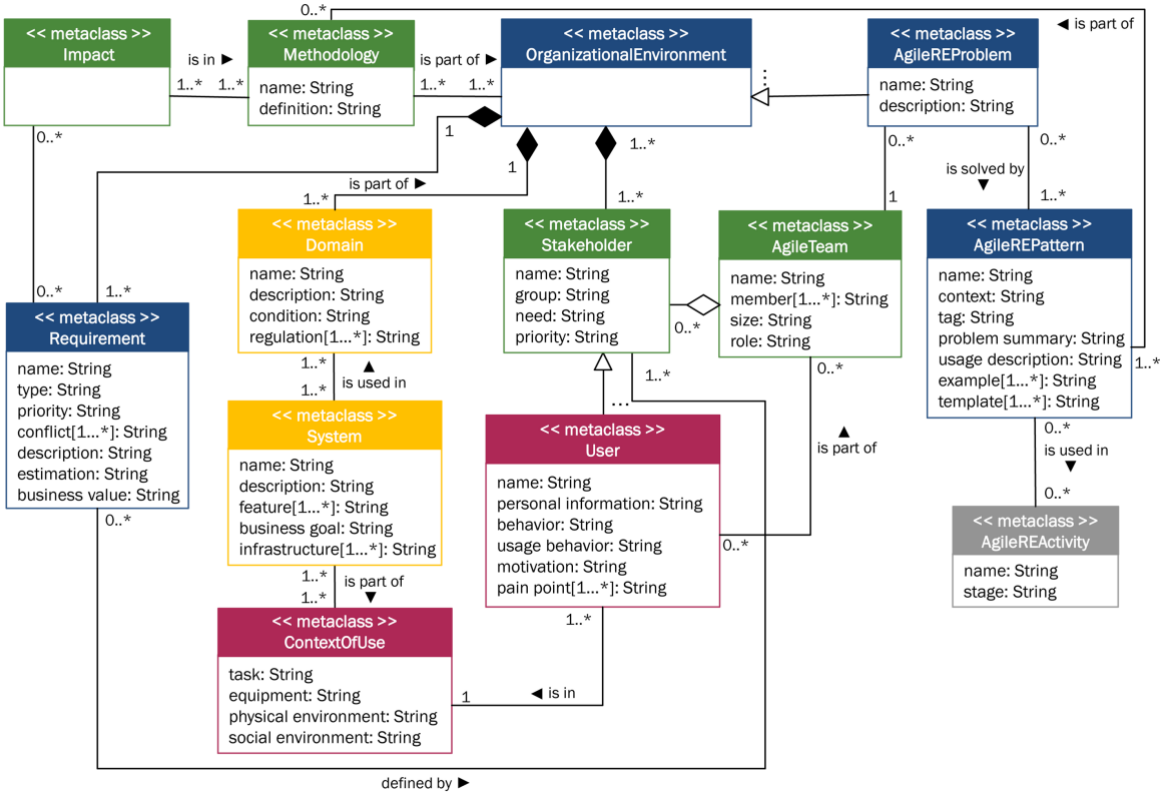


Figure 4: Agile RE metamodel (Schön et al., 2019)

Regarding the commonalities between the proposed metamodels, we can conclude that although they serve different purposes, they contribute to a systematic requirements reuse approach based on metamodels.

2.3 The QDAcity-RE method

The QDAcity-RE method used at our university consists of three main steps that can be processed in iterations (Kaufmann, 2021):

1. Stakeholder sampling
2. Data gathering
3. Data analysis.

Figure 5 gives an overview of the QDAcity-RE process that is used for domain analysis.

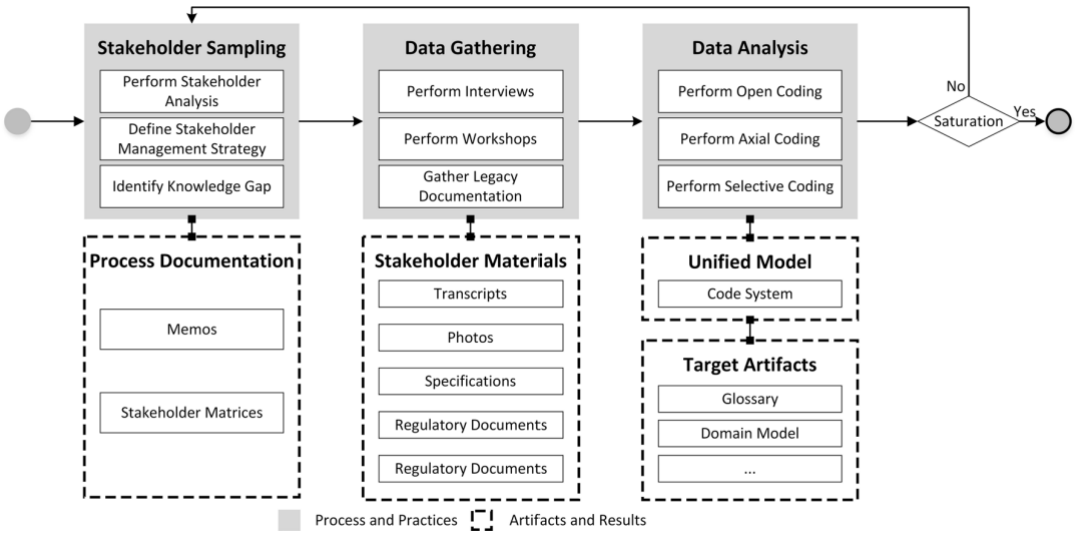


Figure 5: The QDAcity-RE Process for Structural Domain Modeling (Kaufmann & Riehle, 2019)

The method starts with the survey of the interest groups, which identifies which stakeholders are behind them. This stage also aims to define the stakeholder management strategy and identify the knowledge gap.

The second step is about data collection. At the end of the data gathering, there is an extensive collection of information that has to be processed: there are interviews with customers, meetings held, there are minutes and photos, and also legislation.

After that, the third step - the actual data analysis - begins. This is where the QDAcity-RE scientific methodology is mainly applied, where a relatively uniform model is created for data analysis standardization. The model creation is based on a process similar to the coding process of the traditional grounded theory approach. The code system is a hierarchical structure of codes, including concepts, categories, properties, and interactions between codes. The elements of a code system are classified according to the code system language, which main constituents are concepts and relationship codes. (Kaufmann, 2021)

The created uniform model is used, for example, to create a domain model in the form of a UML class diagram. (Kaufmann et al., 2022). Finally, for the third step of the QDAcity-RE process, we have tool support, namely, support through the QDAcity application, which was developed by Kaufmann (2021) as part of the dissertation.

One of the main benefits and expected advantages of the QDAcity-RE method is traceability between the original documents, like stakeholder interview transcripts, and higher adaptability in case of changes in the textual base of requirements. (Kaufmann & Riehle, 2015)

2.4 Preliminary work

We base our work on a preliminary work of Salow in 2016 at the Professorship for Open Source Software which suggested a metamodel for code systems, as shown in Figure 6. This model has three main constituents: labels, aspects and relationships. The labels comprise categories, concepts and properties. Furthermore, there are two groups of aspects: structural and dynamic. The structural aspects consist of objects, actors, and places, whereas the dynamic aspects consist of activities and processes. Finally, the relationships, which connect aspects, are also divided into two groups: structural and dynamic. “Is a”, “is part of”, “is related to” represent structural relationships, and dynamic relationships comprise of the types “is a consequence of”, “causes”, “performs” as well as “influences”.

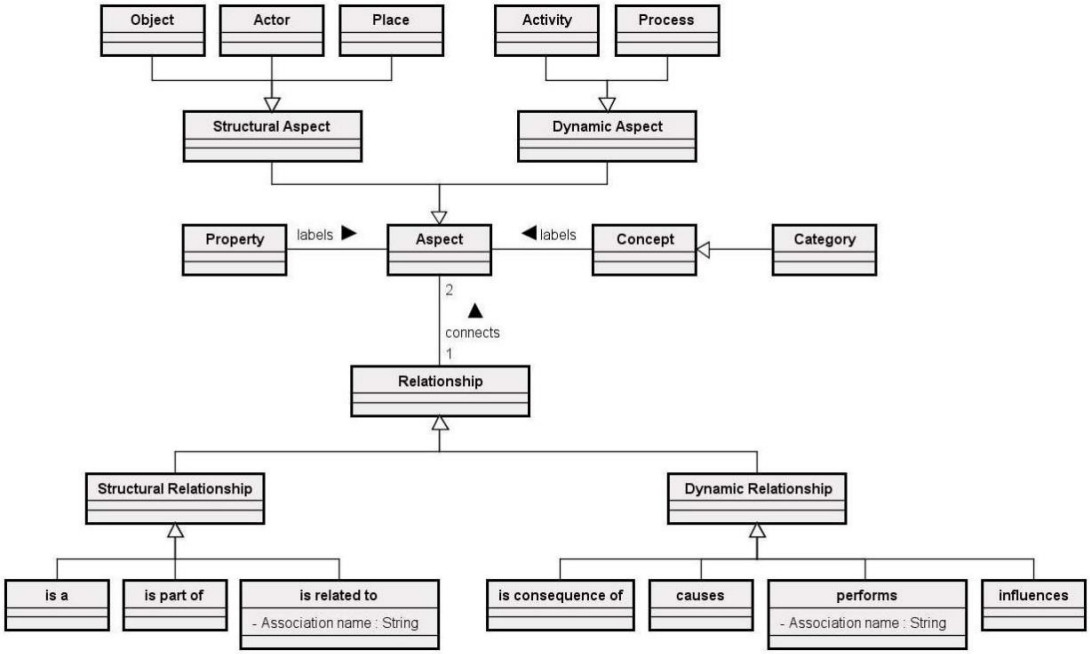


Figure 6: Metamodel, (Salow, 2016)

Due to mapping from a code system assorted with the help of code system language (CSL), UML class diagrams can be derived, and we can transfer the code system metamodel elements to domain model elements. Table 1 shows which CSL elements correspond to which elements in a domain model.

Code System Meta Model	**Domain Meta Model**
Structural category	Class
Property	Attribute
Structural concept	n/a
Dynamic category	n/a
Activity	Operation candidate/association candidate
State	n/a
Is property of	Is attribute of
Is a	Generalization
Is part of	Aggregation
Is related to	Association
Causes	n/a
Is consequence of	n/a
Affects	Indicates operation/indicates association

Table 1: Code System Meta Model Elements transferred to Domain Meta Model Elements (Kunz, 2015)

3 Research design

To reach the goal of creating a metamodel that supports the requirements specification creation process, we firstly started by analysing the stakeholder statements from the interviews in the HR area. Then, when we applied the QDAcity-RE methodology, we first went through all the texts, Figured out significant text segments that were important for later requirements specification creation, and annotated these with codes we defined ourselves. These codes are collected in a list.

After the open coding process, we sorted the created list hierarchically. For that purpose, we also created more codes so that others could be grouped, which gave us the opportunity to create a relatively clear order. After structuring the code hierarchically, we created a code memo for each code, which contained the description, specific information about the code and, if needed, important notes.

This approach has the great advantage that if we have the specification later and have to look up what it is, e.g., "onboarding" or where it comes from, we can use this code to access the code system and even customer statements. That means we can access more contextual information. For example, we can also see how many stakeholders have expressed about the particular code. In addition, we can look at how many codes we have already used in the specification and identify if there are codes in the code system that we have not used in the specification that might be missing.

The next step is to derive a requirements specification from the code system. Therefore, we decided to take sentence templates from a classical requirement and replace the elements of the sentence template with the codes from the code system. We used the sentence template from SOPHIST. Figure 7 represents the template in English, and Figure 8 shows the German version of the same sentence template.

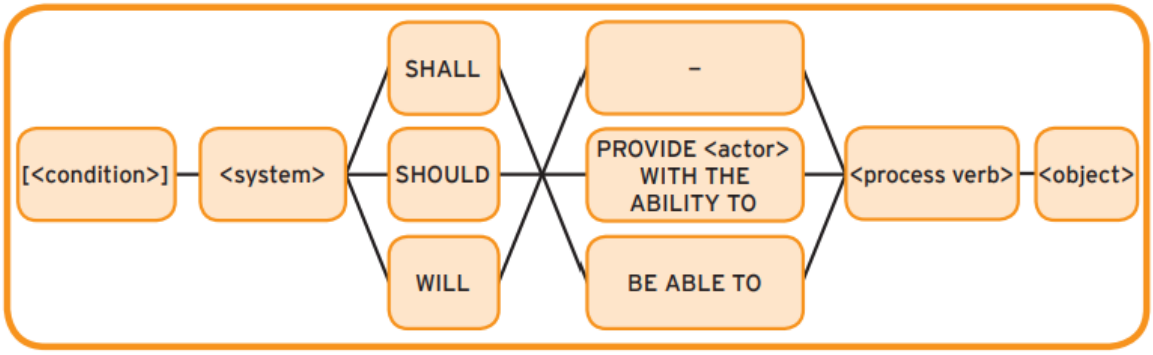


Figure 7: SOPHIST template - English version (Kluge & SOPHIST GmbH, 2016)

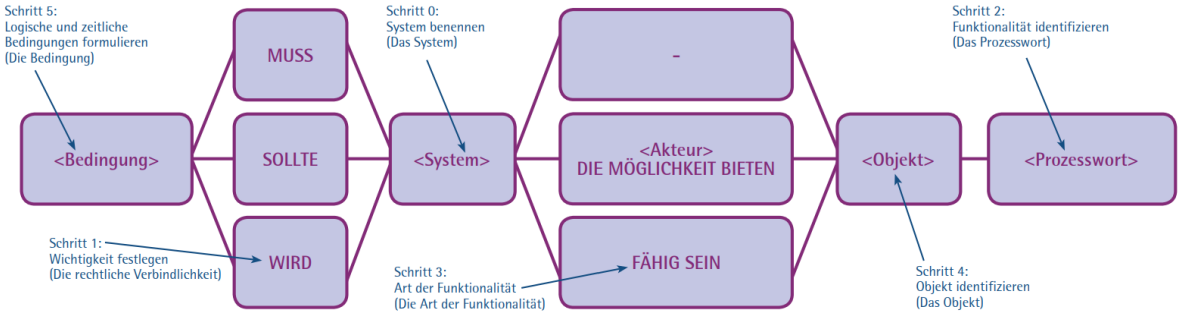


Figure 8: SOPHIST template - German version (SOPHIST)

Based on the code system and with the use of the template, we manually created sentences for the software requirements. During the process of the specification creation, the method of peer debriefing was used when the first version of the requirements specification was created. This technique is used to improve the validity of research and is applied in qualitative research, particularly in grounded theory. (Barber & Walczak, 2009) The latest version of the specification was adjusted, taking into account the peer debriefer's – the thesis supervisor's - feedback based on the critical review.

3.1 Research question

This thesis aims at analysing and refining the existing metamodel for domain analysis to support the creation of a natural language requirements specification. The research questions are:

1. Is it possible to use the same underlying code system to create a domain model and a requirements specification?

1.1. If not, what are the differences?

2. Is it possible to use the same metamodel to create a domain model and a requirements specification?
3. How can a metamodel support the creation of a domain model and a requirements specification?

4 Used data

In this thesis, six interviews on HR development were used as a base for our research. The interviews were conducted by Kunz (2015) and were in German. For this purpose, four domain experts from different companies were interviewed, two of whom additionally took part in follow-up interviews for further clarification. The companies where the domain experts worked differed in their sizes and levels of digitization. The domain model was created based on the code system as a UML class diagram.

For our research, we used the code system introduced by Kunz based on the data set of the six interviews for comparison to identify the similarities and differences.

5 Research approach and results

5.1 Analysis of the interviews

Each type of code we used for coding the stakeholders' interviews is marked with a specific colour, as presented below in Table 2. To make the process more convenient, we used colours to highlight the codes belonging to the same metamodel element.

Metamodel elements	Colour
System	
Actor	
Process (Code including process verb + Qualifier (Should/Sollte, Will/Wird, n.a.))	
Process (Code including process verb + Qualifier (Shall/Muss))	
Groups or other codes	
Condition	
Addition/Supplement	
Not relevant information for requirements specification	

Table 2: Metamodel elements and corresponding colours

The code system is equivalent to the system from the sentence template described in chapter 3. We used the root code of the code system - created automatically by default when the coding editor of QDAcity is opened to start coding the interviews - as a code.

An actor is a noun, e.g., "employee". Processes are performed by an actor.

A code, including process verbs, needs to be represented in requirements specification and therefore has to be explicitly coded. For example, the code "document the results of the conversation" ("Gesprächsergebnisse dokumentieren") has a significant meaning and will be included in the requirements specification of the software for HR development. Therefore, it is coded explicitly and labelled in green colour. This code also includes an object: "the results of the conversation" ("Gesprächsergebnisse").

The qualifier determines the relevance of the requirement. Depending on the binding character, there are three types of requirements: "shall", "should", and "will". (Kluge et al., 2016) If not explicitly coded as such, a "should" requirement is further used in our specification. If the code is legally binding or strongly necessary, then the code is marked as red to gain attention to the codes in this colour. The "should" requirements and the ones which binding character is not precise are marked with green in the code system. In our case, we did not explicitly code "will" requirements. However, in the future, this type of code can be marked in another colour. An example of a "shall" requirement is based on the following stakeholder statement: "With us, the employees can simply register for the training courses, and the superiors *must* approve this in the system that we have for it". In this case, a code "supervisor approves" ("Vorgesetzter genehmigt") was created, and as in the stakeholder's interview was stated "the superiors *must* approve", we marked that code with red.

Concepts can be grouped into categories. (Corbin & Strauss, 1990b). For example, the codes "organization" and "goals" are parent codes, which can have zero coded text segments as they serve as group codes. These categories represent a coding unit and can be described in a corresponding code memo. Other codes, labelled in black, usually represent objects. For example, the codes "qualification profile" ("Qualifikationsprofil") and "goals" ("Ziele") represent the metamodel element object.

The condition can be either a temporal or logical condition. An example of the temporal condition is "at the end of the year" ("am Jahresende"), and an example of a logical condition is "if the maximum number of applicants is exceeded" ("falls maximale Anzahl der Bewerber

überschritten”).

The metamodel element addition/supplement represents codes that provide us with additional information that can be relevant to requirements. Examples of such codes are “by email”, “via intranet” and “as an excel file” (“per Email”, “über Intranet”, “als Excel Datei”).

The codes in grey are not relevant to the sentence template of a requirement. However, they deliver some information about the processes or can serve as a better understanding of the actual situation and are relevant for context information. The “company size” and “position in the company” of an interviewee are examples of irrelevant codes for the sentence template of a requirement.

5.2 Comparison of code systems

We performed qualitative data analysis of the data on the same interview documents as Kunz used to create the domain model. However, we used these documents intending to create a requirements specification. We coded the interviews without looking at the former code system for domain analysis to ensure the independence of the coding process and to determine the similarities and differences between both code systems. Chapter 5.2.1 shows the quantitative aspect of the comparison, whereas chapter 5.2.2 deals with the qualitative part. Further, we will refer to the two code systems as the code system for domain modelling (see appendix A) and the code system for requirements specification (see appendix B).

5.2.1 Quantitative comparison

In the code system for requirements specification, we have defined 151 codes compared to the code system for domain modelling, which contains 125 codes, as shown in Figures 9 and 10. In our case, we have 320 coded text segments in total.

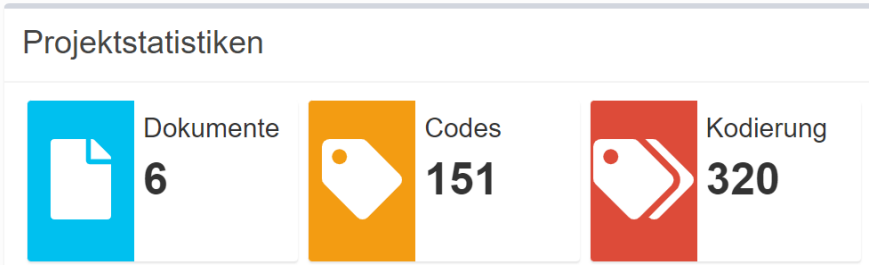


Figure 9: Project statistics from QDAcity

On the contrary, the other code system, designed for domain analysis, contains 555 codings. It also includes 128 Memos, 124 of which were code Memos, and the rest were referred to as “free Memos” that had no direct connection to the codes of the code system and contained general notes. These free memos were assigned neither to the codes nor to the documents. Instead, they were connected to the whole data set.

Codes: 125
Codings: 555
Codesets: 0

Memos: 128
Dokument-Memos: 0
Memos in Dokumenten: 0
Code-Memos: 124

Figure 10: Project information from MAXQDA

One of the codes, named “Performance management cycle”, refers to the picture with the outline of the process management cycle. Therefore, the 554 codes refer to the six interview transcripts. Figure 11 shows the list of all documents in the data set and their corresponding numbers of codings to illustrate the distribution of the codings on the individual interview documents.

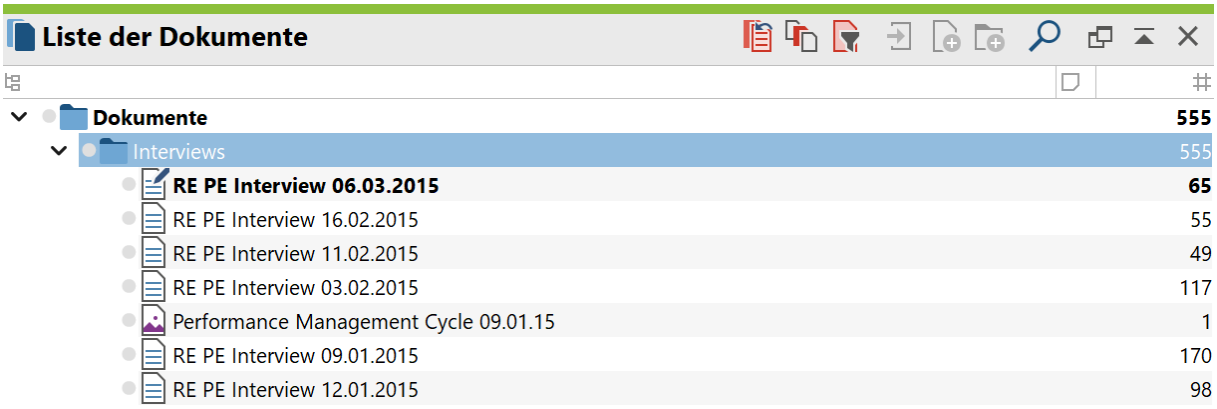


Figure 11: List of the documents in MAXQDA

Figure 12 shows the number of codings for each interview in both code systems and their differences. The code system for domain modelling contained more coded text segments in all the interviews. The most significant difference was found among the numbers of coded segments in the third interview, and the minor difference was detected in the fifth interview.

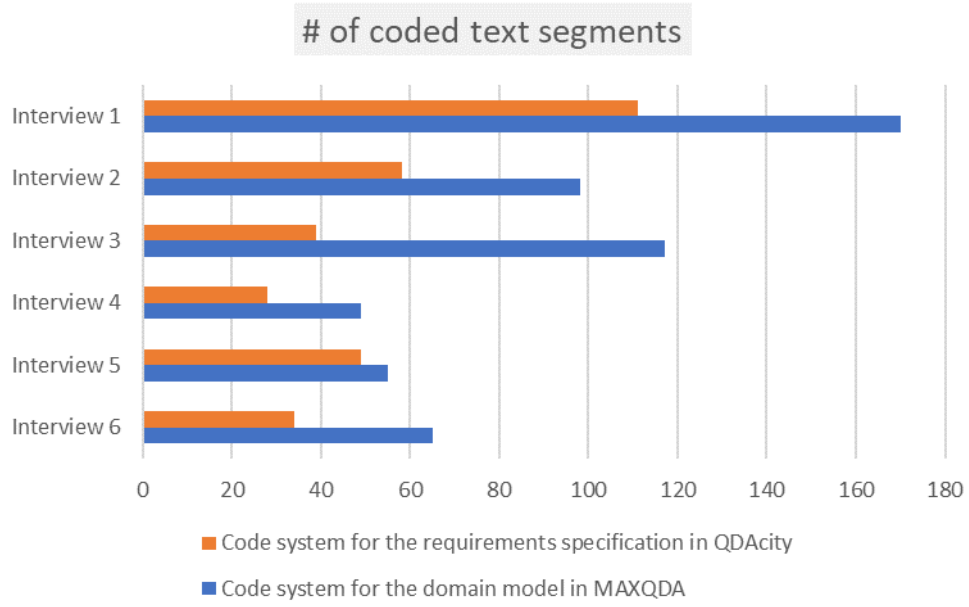


Figure 12: Comparison of codings by the number of coded text segments

As described in chapter 5.1, Analysis of the interviews, different colours were used to differentiate various codes. Figures 13 and 14 present below the total numbers of each type of code for both code systems.

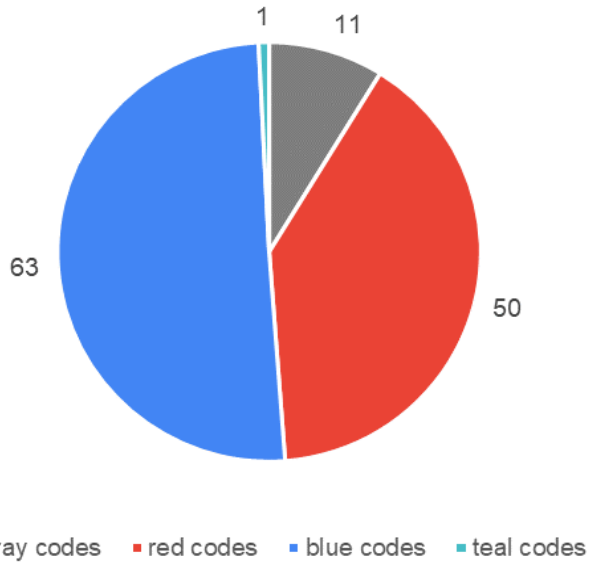


Figure 13: Colours of codes in the code system for the domain model in MAXQDA

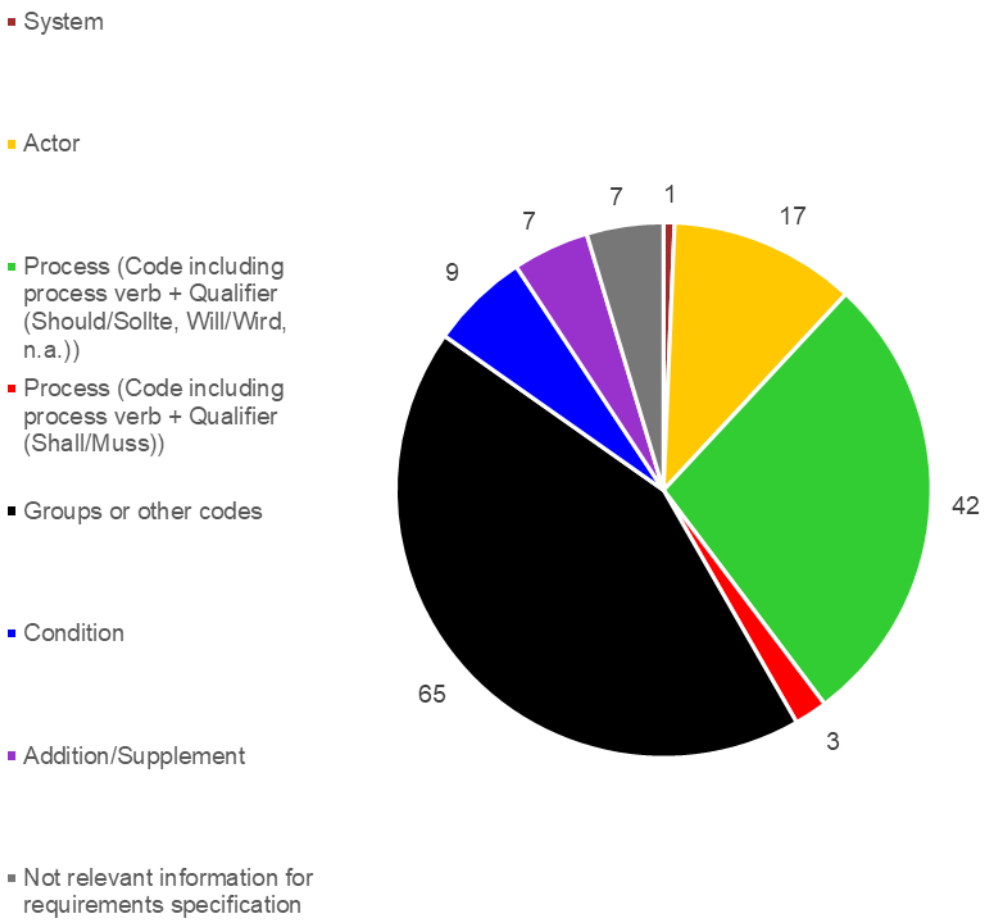


Figure 14: Colours of codes in the code system for the for requirements specification in QDAcity

The code system for requirements specification and the code system for domain modelling can also be differentiated by the number of hierarchical levels presented in Figure 15. Whereby the code system for requirements specification consists of ten levels - from 0 to 9, the code system for the domain modelling has six hierarchical levels. The hierarchical level 0 is represented by the automatically generated code “Code System” in QDAcity, whereas in MAXQDA, it is not counted as an explicit code.

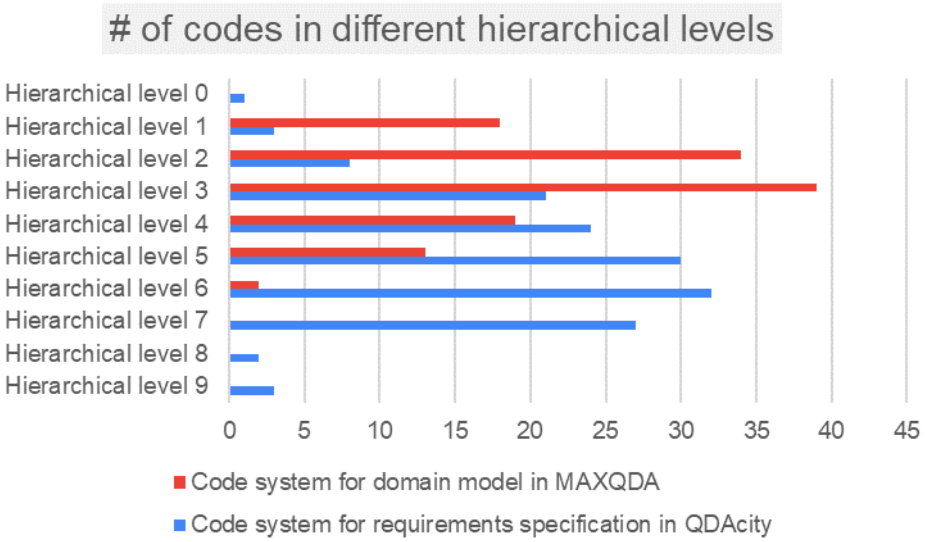


Figure 15: Comparison of code systems by the number of codes in different hierarchical levels

5.2.2 Qualitative comparison

The code system for domain modelling was coded with the MAXQDA application, whereas we developed the code system for requirements specification with the help of QDAcity. Both applications are designed for qualitative data analysis.

In our case, we only used actors as structural aspects and did not code objects and places separately. As we coded intending the creation of a specification, no places were found to code to be further included in the requirements. The objects were either included in the codes that contain the process verb or in the black labelled codes as “other codes”. Compared to the code system for requirements specification, the memos assigned to the codes of the code system for domain modelling included information about the metamodel elements. They are called concept types in those memos and are used synonymously to metamodel elements. Examples of codes of a concept type object are “interview guide” (“Gesprächsleitfaden”), “social skills” (“Soziale Kompetenzen”) and “development needs” (“Entwicklungsbedarf”). None of the text segments was coded as a place neither in the code system for domain modelling nor in the code system for requirements specification.

In the code system for domain modelling, examples of activity are “evaluation feedback” (“Auswertung feedback”) and “gather feedback” (“Feedback einholen”). However, “evaluation” and “gather” represent different parts of speech. “Evaluation” is a noun, whereas “gather” is a verb. Another pair of examples are „comparison with colleagues” (“Vergleich mit Kollegen”) and „approve development measures” (“Entwicklungsmaßnahmen genehmigen”). Also, here, “comparison” and “approve” belong to different word categories and are not named consistently.

Kunz’s colours differ from those we used in our coding. We found the partial meaning for this colour selection in the description of the “free memo” s called “coding method” (“Kodiermethode”) in the data from MAXQDA. There was found the following note, translated

from German: “Quotation marks stand for in-vivo codes, blue codes for background knowledge/framework conditions. Text passages often have to be assigned multiple codes. If I only code the individual words, the context is lost. Gray codes may be irrelevant.” Unfortunately, there was no information about the meaning of codes in red and teal. As Figure 13 shows in the previous chapter, about half of the codes were labelled with blue colour, which represented background knowledge and framework conditions. There was only one code in the teal colour.

In the code system for requirements specification, we used the metamodel element condition with another meaning than Kunz. Furthermore, we used either logical or temporal conditions, for example, "at the end of the year" and "if the maximum number of applicants is exceeded". On the contrary, in the code system for domain modelling, the "position of the interviewee" was defined as a condition in the corresponding memo ("Position des Interviewten"). Therefore, it belonged to the description of the actual situation in the interviewee's company.

As shown in the previous chapter, we have many codes in the code system for requirements specification. That might be due to the larger number of hierarchical levels. Also, we have defined more group codes than the code system for domain modelling. Partially they stand for better order and clarity in the code system for requirements specification and in our case, they do not have any connected coded segments in the interviews. That can also justify the higher number of codes in our code system.

5.3 Creation of specification

We coded the interviews in German and then created the requirements specification in German. As Kunz's code system was also in German, we could directly compare both code systems without translating. The process of the specification creation was manual based on the code system. During requirements' creation, each requirement was manually given a sequential number which consists of a prefix, a counter and a brief name of the requirement (see appendix C). We created those sequential numbers to enable a clear structure and effortlessly sort the requirements within the software requirements specification. For example, the prefix and counter of the first requirement name are "Req. 1:" and the prefix and counter for its sub-requirement are "Req. 1.1:". The name of each requirement is a noun derived from the corresponding codes linked to the requirement.

Table 3 represents the six steps that lead to a requirement's creation based on the template of SOPHIST.

Step	Action
1	Name the system (The system)
2	Determine the importance/the legal liability of a requirement
3	Identify the functionality/the process word
4	Determine the type of functionality
5	Identify the object
6	Formulate logical and temporal conditions

Table 3: Steps for creating a sentence based on the SOPHIST template

As described in the previous chapter, the code “system” is equivalent to the system from the sentence template. It is replaced by “HR-Software” in our software requirements specification.

The second step is about the determination of the importance and the legal liability of a requirement. In our specification, we only used shall- and should-requirements as, in our case, we did not have any requirements that would correspond to the will-type of legal liability.

Then we first have to determine the core of the requirement, the desired functionality or the executed process. This should be described as precisely as possible by complete verbs as they are highly relevant for a requirement. Therefore, the codes, including the process verb, are fundamental units of the code system for requirements specification. While formulating a requirement, we write down this process verb at the end of the requirement sentence in German.

There are two types of functionality according to the SOPHIST’s template presented in Figure 16: “provide <actor> with ability to” and “be able to”. We only needed the first type in our requirements specification to link actors to the functionalities described in the requirements.

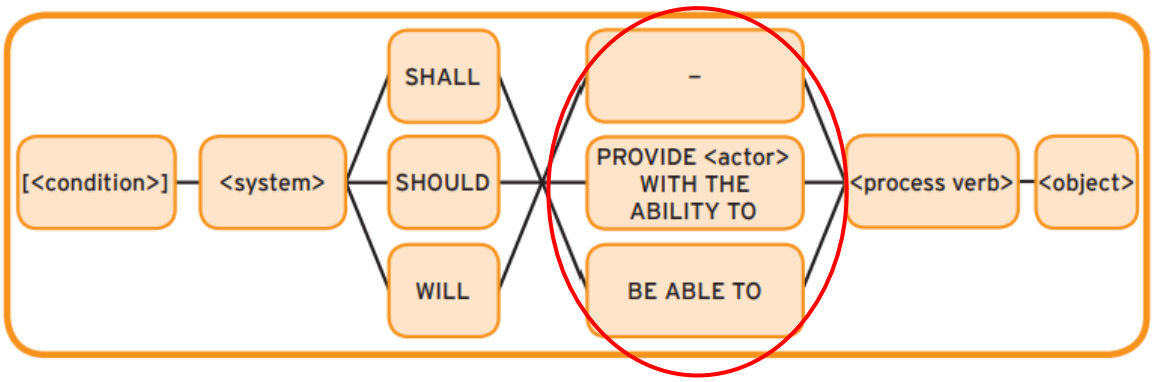


Figure 16: FunctionalMASTER (Kluge & SOPHIST GmbH, 2016)

During the specification creation, we linked the actors to the corresponding requirements. For example, as shown in Figure 17, "The HR software provides the participant with the ability to fill out the feedback questionnaire." In this case, we linked the code "participant" ("Teilnehmer") to the requirement.

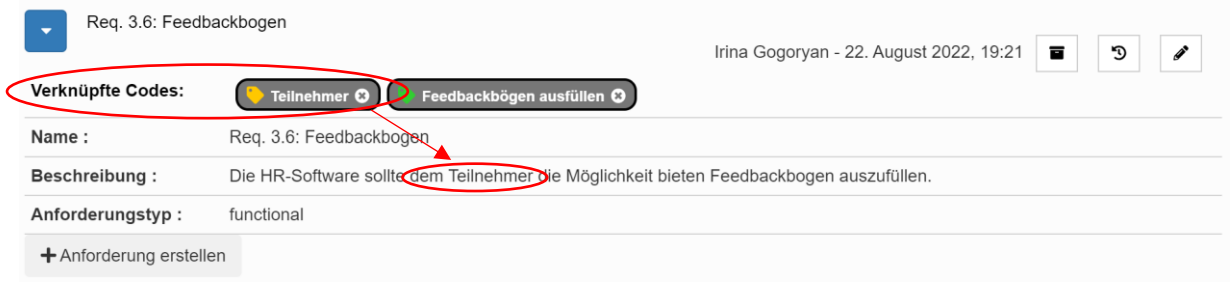


Figure 17: Requirement 3.6 - Screenshot from the requirements specification in QDAcity

We tried to link the actors to the corresponding requirements as precisely as possible. However, when interviewees did not specify actors in their interviews, we used a code from a higher hierarchy level for an actor to connect to the requirements in the specification. For example, in Req. 2, named "evaluation of employees" ("Evaluation der Mitarbeiter"), we linked the group code "actors" ("Akteure") to the requirement, as shown in Figure 18.

"You try to get 360 feedback about people by asking colleagues, department heads, HR or customer side." The latest is a sentence from one of the interviews, which shows that external actors, like the customers, can also participate in the employee evaluation. Therefore, we wrote the requirement in the following way: "The HR software should provide the user the ability to evaluate an employee.".

Figure 18: Requirement 2 - Screenshot from the requirements specification in QDAcity

In case it was clear that the user should be an internal actor, but the interviewees revealed no further details about the exact actor in the interview, we linked the group code internal actors ("interne Akteure") to the requirement, as shown in Figure 19. We also did so when we were unsure about the information about the actor. ("The HR software should provide the internal user with the ability to measure the success of the training."). If we needed clarification on a specific actor, we wrote a note in memos connected to the coded text "ask the customer for

further information".

Figure 19: Requirement 3.11 - Screenshot from the requirements specification in QDAcity

We used the terms user, internal user and external user within the requirements correspondingly for the codes actors ("Akteure"), internal actors ("interne Akteure") and external actors ("externe Akteure").

The next step is the identification of the object. An object is a noun. In the code system for requirements specification, we usually coded the process verb together with an object in order to keep the code system precise. However, if, in the future, the automatic creation of the requirements specification with the help of the QDAcity application is technically possible, it will be more meaningful to code all the process verbs and objects separately.

The last step of the requirement's creation is formulating the logical and temporal conditions. Examples of logical conditions used in our requirements specification are "if the employee is new" ("falls Mitarbeiter neu ist") and "if there are any changes" ("falls es Änderungen gibt"). "During the year" ("während des Jahres") and "at the end of the year" ("am Jahresende") represent examples of temporal conditions.

The code system for requirements specification has a code type addition/supplements, for example, "via email" ("per Email"). If we write in English, the additional information is placed at the end of the sentence right after the process verb. An example for such requirement is: "The HR software provides the internal user with the ability to communicate the appointment to the employee via email." („Die HR-Software bietet dem Nutzer die Möglichkeit dem Mitarbeiter den Termin via Email mitzuteilen.")

When the sentence is already created and if a user is unknown, we link the parent code "actors" ("Akteure") to the requirement and try to determine the user in future interviews with the customers. In case we know that the user is someone who has been hired by the company but are not aware of more detailed information about the user's position and functions, we use the group code "internal actors" ("interne Akteure"). Furthermore, if a user is not working at the company, we use the code "external".

In QDAcity, we can select between the options functional and non-functional requirements. An example of a non-functional requirement is "The HR software provides the workers' council with the ability to observe the data protection law.".

Suppose there is a condition in a sentence. In that case, we put the condition in the first place of a sentence, based on the template in Figure 20, as in the following requirement: "If it is a virtual training, the HR software provides the operating department and the HR department with the ability to check the target group."

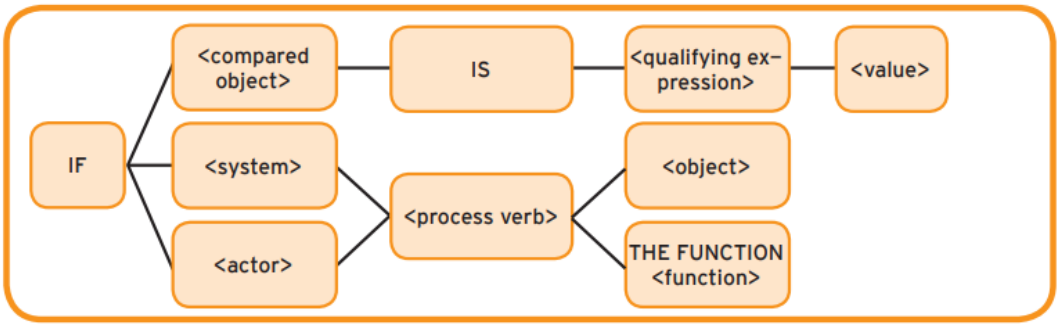


Figure 20: FunktionsMASTER with conditions (Kluge & SOPHIST GmbH, 2016)

A functional requirement can also be formulated without a condition. In this case, the structure of a sentence will be changed in German.

Regarding the completeness of the requirements specification, we tried to use all the codes in the code system. Nonetheless, we could not use all of them, as they either contained background information or we needed to clarify their interconnection with software requirements in further interviews. Figure 21 shows how many codes from the code system for requirements specification were involved in the specification and how many were not used. All the shall-process codes, conditions and additions/supplements were involved in the requirements specification. For example, the code "education" ("Ausbildung") is mentioned in one of the interviews as part of the HR development tasks. However, no processes regarding this code that needed software support were described, which is why it is not included in the requirements specification.

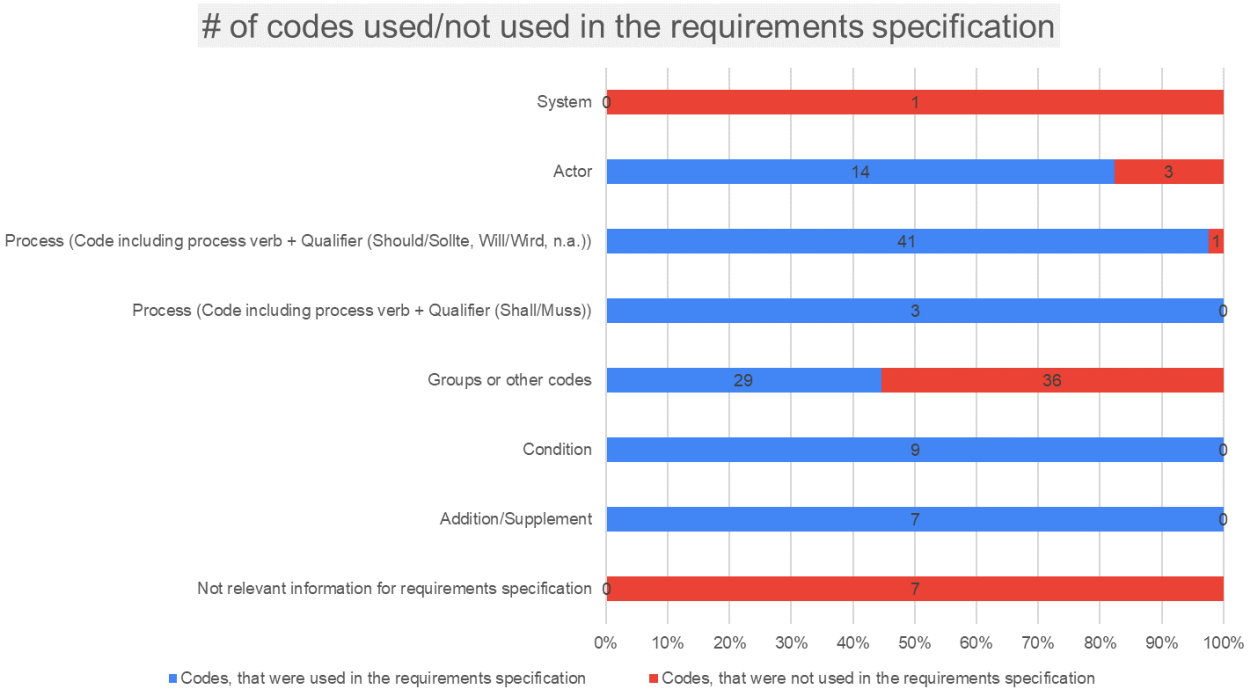


Figure 21: Completeness analysis of the requirements specification

5.3.1 Challenges in creating a specification based on the code system for domain modelling

Trying to create a requirements specification based on the code system for domain modelling, we found that some information was missing or should be adjusted.

For example, we need to check if the code “comparison with colleagues” (“Vergleich mit Kollegen”) is a process, event, or rather an activity. For this purpose, we need to open and read the corresponding memos. Consistent usage of the word category verb would be better in this case.

The inconsistent form of coding the activities increases the workload of an analyst, who aims to create a specification based on the code system. Therefore, we decided to adjust the existing metamodel to be able to derive a requirements specification.

5.4 Proposal for a unified metamodel

In order to meet the goal of creating the requirements specification, we extended the existing metamodel by Salow, which can be seen in the Figure 6. Our proposed metamodel is presented in Figure 22. It is reusable and can be applied to other data sets.

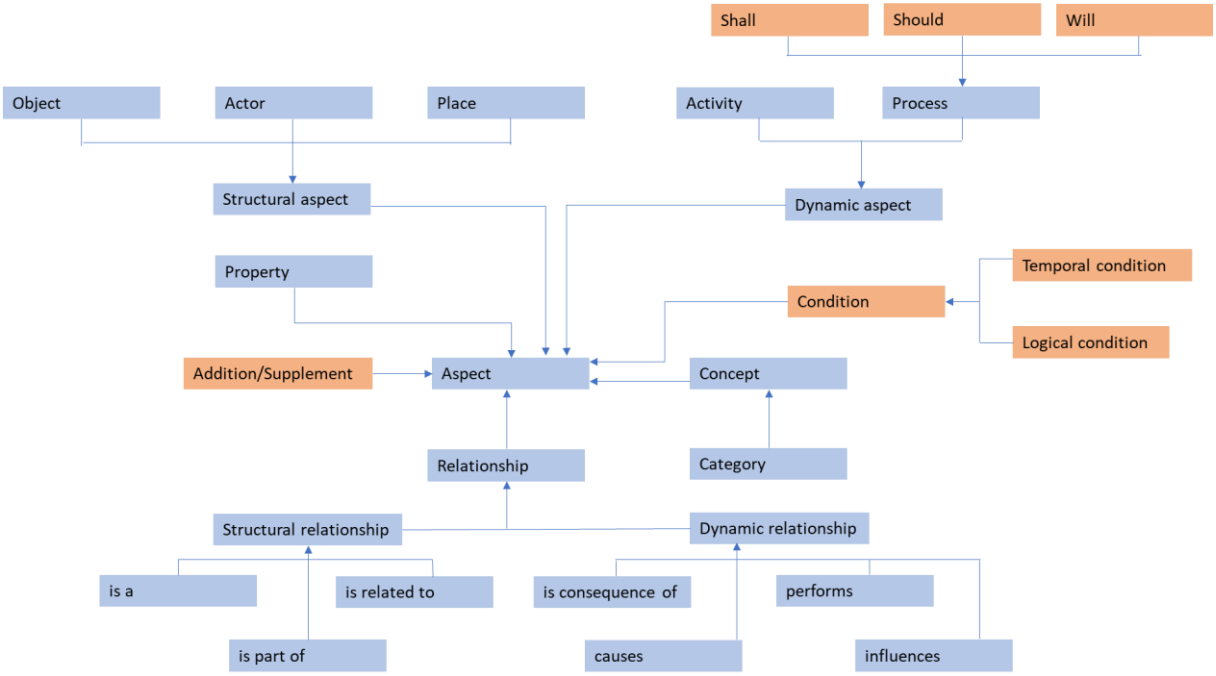


Figure 22: Unified metamodel

We considered that in the best case, it should be possible to use the same metamodel to create a domain model and a requirements specification to reduce the workload of an analyst and the storage space, avoid redundancies and save time. Therefore, we decided to adjust and extend the existing metamodel to support the requirements specification creation without eliminating any metamodel element that was not used during the requirements creation. An example of that is the "is a" relationship. It is applicable for description, for domain information, however, this information will not be included in the requirements, as we try to keep them as precise as possible. Another example is the property that describes an aspect. The code "type" ("Art") from the code system for requirements specification was subordinated to the code "competencies" ("Kompetenzen"). Although "type" is relevant to the domain model as a property, we did not include it in the requirements.

We extended the metamodel for domain modelling with metamodel elements depicted in orange in Figure 22. The metamodel element condition can be classified as a temporal or logical condition. It clarifies under which conditions the software should perform specific functionality.

Another added metamodel element - addition/supplement – shows the codes containing additional information relevant to requirements. For example, the codes with this type of metamodel element can give us details on a process.

As one of the core parts of a requirements sentence template, the process verbs are encompassed in codes belonging to the metamodel element process. It is essential to pay attention to a consistent way of coding, especially the wording used to code the processes, not to make creating requirements sentences more difficult and directly use the process names while building the sentences. A process can be classified as shall-, should- or will-type depending on the level of relevance. (Kluge et al., 2016) If the analyst is unsure about which level of relevance is to assign to the process, then the metamodel element "should" is to be used for coding.

The colour labels can be further used to support the stakeholder interviews analysis phase, as

described in chapter 5.1. The adjusted table consisting of metamodel elements and corresponding colours is presented in Table 4. The colours can help keep track of the codes that seem essential to the analysts. For example, the processes with the highest level of relevance get attention due to the red colour of the code and build a basis for shall-requirements.

If needed, temporal and logical conditions can also be labelled in different colours. However, as they have the same position in a requirements sentence, one colour can be sufficient for labelling both types of conditions.

Metamodel elements	Colour
System	
Actor	
Process (Code including process verb + Qualifier (Should/Sollte, n.a.))	
Process (Code including process verb + Qualifier (Shall/Muss))	
Process (Code including process verb + Qualifier (Will/Wird))	
Groups or other codes	
Condition	
Addition/Supplement	
Not relevant information for requirements specification	

Table 4: Adjusted table for metamodel elements and corresponding colours

6 Discussion

While adjusting the metamodel, there has arisen a question of whether a process verb could be better addressed by a process code or a relationship between codes. We suggested involving the processes in codes because this made it easier to create sentences in the specification editor of the QDAcity tool only by looking at the code system without separately checking all the metamodel-specific information stored behind the codes. We only checked the information in questionable cases. However, in this thesis, we did not analyse whether our approach leads to a better quality requirements specification. The question of whether a process could be better expressed by a process code or a relationship between codes and overall, the application of the metamodel should be analysed in further research.

For further validation of the metamodel, we suggest applying the metamodel to new data sets using the following procedure:

1. An independent analyst should create an ad-hoc specification without using a meta-model.
2. Another independent analyst should create a requirements specification making use of the code system based on the metamodel using the same data set.
3. This ad-hoc requirements specification should be compared to the requirements specification derived from the code system based on the metamodel.
4. Expert feedback should be used for evaluating and comparing both requirements specifications.

Suggestions for the future usage

Apart from the templates for requirements sentences used in this thesis, more detailed versions of templates are available. LogicMASTER, EventMASTER and TimeMASTER templates from SOPHIST can be considered in case of more complex requirements in the future. (Kluge & SOPHIST GmbH, 2016)

Supposing that in the future, automated creation of specifications based on a code system will be possible in the QDAcity tool, it would be interesting to use keywords such as confidentiality, completeness, password, memory and runtime that can help automatically classify non-functional requirements. (Cleland-Huang et al., 2007)

Suggestions for tool features

While working with the QDAcity tool, we came up with suggestions for features in the application to support the processes of coding and requirements specification creation.

In the coding editor of QDAcity, we first coded the interviews to create a requirements specification further. In case coded text segments that included a specific code were not clear enough or had to be clarified with the customer, we wrote a note to ask the customer in the memo of the code. It would be helpful to search for all the codes like this to determine what information is missing for the requirements specification and to create questions for follow-up interviews with the customers. Furthermore, searching for labels of the same colour would be very helpful because we used many different colours during coding. Also, searching for codes of the same hierarchy level could be interesting. Finally, maintaining the sequence of codes in the code system while exporting them from QDAcity to an Excel document would be helpful.

We used the specification editor in the QDAcity application to create the requirements specification. A filter showing functional and non-functional separately or separate sections for different types of requirements could be helpful for more extensive specifications. Moreover, it

could be useful to filter the requirement based on the shall-, should- and will- qualifier and show them in different colours or use specific icons for each level of relevance. Showing all requirements connected to one specific type of actor would make it easier for an interviewer to prepare questions for follow-up interviews with customers. Numbering the requirements and putting them automatically in ascending order would also help create them.

QDAcity has an option to show if a code has a memo or metamodel information assigned to it. Showing some part of the context of the memo and the metamodel information while hovering over the memo and the metamodel symbol situated next to each code would help analysts to save time while working.

7 Limitations

The interviews were held by another person in 2015, and we had no access to the four interviewees. Therefore we could not conduct further interviews to receive more detailed information in terms of the creation of the requirement specification and to clarify misunderstandings. For example, if we had limited information regarding the actors that should be linked to a specific requirement, we used a code from a higher hierarchy level, e.g. the code “internal actors”.

Nonetheless, even if we conducted follow-up interviews, the actual situation should not have been changed a lot so that it could influence and bring significant changes in the code system for requirements specification. On the other hand, in some cases, having more detailed information, we could formulate the sentences in the requirements specification more precisely.

We only focused on developing requirement sentences. However, the recommended structure for a good software requirements document includes more information and sections, as shown in Figure 1.

Although use cases and user stories are widely used in many companies to document requirements, we did not use them during our research. Instead, we documented the software requirements in a natural language specification in the QDAcity application.

In terms of completeness, not all the codes in the code system for requirements specification were involved in the requirements specification.

Regarding validity, we only tested our metamodel on one data set.

8 Conclusions

This thesis aimed to analyse and adapt the existing metamodel to support the creation of a natural language requirements specification. The refinements of the metamodel should allow the creation of a conceptual domain model and a natural language requirements specification.

To meet the goal, we defined three research questions at the beginning of this work in chapter 3.1. In order to answer the first research question, whether it is possible to use the same underlying code system to create a domain model and a requirements specification, we first started analysing the interviews with stakeholders by applying the qualitative data analysis method, particularly the QDAcity RE method. As a result, we developed a hierarchically structured code system for requirements specification and compared it to the code system for domain modelling from preliminary work. Then, trying to derive a requirements specification from the code system for domain modelling, we found that some information, such as conditions, was missing, or the metamodel should be adjusted so that we could replace the elements of a standard sentence template for requirements with the corresponding metamodel elements in order to form a natural language requirements specification.

Our finding also answered the second research question, if it is possible to use the same metamodel to create a domain model and a requirements specification. Creating a requirement sentence based on the underlying information from the code system for domain modelling could still be partially possible. However, the requirements would not be precise enough and complete. They would not contain conditions, for example, and the art of the binding type so that we could not differentiate the shall-, should-, and will- requirements. Moreover, the core of a sentence, a verb, should be defined as precisely and consistently as possible, as a sentence would not make sense without a good process verb. As a result, we presented a refined metamodel. We decided not to create a new metamodel specifically for a requirements specification. We only extended the existing metamodel, as removing an element could hinder the creation of a domain model. Therefore, a domain model and a requirements specification can be derived based on the same refined metamodel.

The last research question was about how a metamodel can support the creation of a domain model and a requirements specification.

As the metamodel is reusable and can be applied to other data, we expect more efficient creation of a software requirements specification to reduce the failure in later phases of software development compared to the ad-hoc creation approach without using the metamodel.

In conclusion, we suggested steps for further validation of the metamodel, which is extended with several metamodel elements, by applying the metamodel to new data sets.

Appendix A - The code system for domain modelling

* L = hierarchical level

L1 • Leistung
- L2 • "Leistungsbeurteilung"
  - L3 • "Projektbeurteilung"
  - L3 • "Kompetenzeinschätzung"
    - L4 • "Selbsteinschätzung"
    - L4 • "Feedback"
      - L5 • Feedbackfragebogen
      - L5 • Feedback einholen
      - L5 • Auswertung Feedback
      - L5 • "360Grad Feedback"
  - L3 • Vergleich mit Kollegen

L1 • Kompetenz
- L2 • Ausprägung
- L2 • Beschreibung
- L2 • Methodenkompetenz
- L2 • Fachkompetenz
- L2 • "Sozialkompetenz"

L1 • Qualifikation
- L2 • "Qualifikationsprofil"

L1 • "Jobcluster"
- L2 • "Rolle"
  - L3 • "Anforderungsprofil"
- L2 • "Karrierepfad"
- L2 • "Gehalt"

L1 • "Performance Management Cycle"
- L2 • "Mitarbeitergespräch"
  - L3 • Datum
  - L3 • Monitoring von Mitarbeitergesprächen
  - L3 • "Gesprächsleitfaden"
  - L3 • "Zielvereinbarungsgespräch"
    - L4 • Ziele des Unternehmens

- L4 • "Ziele des Mitarbeiters"
- L4 • "Ziel"
  - L5 • Zeitraum
  - L5 • Entwicklungsziel
  - L5 • Leistungsziel
    - L6 • Messgröße
    - L6 • Zielwert
- L3 • Evaluationsgespräch
  - L4 • Zielevaluation
    - L5 • Zielerreichungsgrad
    - L5 • "Variable Vergütung"
  - L4 • Endjahresevaluation
  - L4 • "Halbjahresevaluation"

L1 • "Entwicklungsbedarf"

- L2 • "Rollenwechsel"
  - L3 • "Beförderung"
  - L3 • Zusatzaufgabe
- L2 • Vorschlag der Führungskraft
- L2 • Mitarbeitervorschlag
  - L3 • "Kompetenzteam"
- L2 • Marktanforderung

L1 • Entwicklungsmaßnahme

- L2 • Teilnahme
  - L3 • Status
  - L3 • Organisation von Entwicklungsmaßnahme
    - L4 • Mitarbeiter informieren
    - L4 • Mitarbeiter meldet sich für Maßnahme an
    - L4 • Entwicklungsmaßnahme genehmigen
    - L4 • Entwicklungsmaßnahme evaluieren
  - L3 • Monitoring geplanter Maßnahmen
  - L3 • Auswertung über belegte Maßnahmen
  - L3 • Kosten
  - L3 • Datum
- L2 • gesetzte Entwicklungsmaßnahme
  - L3 • Veranstaltung für neue Mitarbeiter

- L2 "Erfolgsmessung von Entwicklungsmaßnahmen"
  - L3 "Return on Investment"
- L2 Methode
  - L3 Übung
  - L3 "Coaching"
  - L3 Mentoring
  - L3 Kollegialer Wissensaustausch
    - L4 Unternehmensweite Veranstaltungen
  - L3 Selbststudium
  - L3 "Externe Ausbildung"
  - L3 Schulung
    - L4 "Schulungskatalog"
    - L4 Anbieter
      - L5 Interne Schulungen
      - L5 Externer Anbieter
      - L5 Corporate University
    - L4 "Virtuelle Schulung"
    - L4 "Präsenzschulung"
      - L5 Trainer
- L2 Entwicklungsprogramm
  - L3 "Traineeprogramm"
  - L3 "Führungskräfteentwicklungsprogramm"
    - L4 Auswahl der potentiellen Führungskräfte

L1 Akteure

- L2 "Benutzerrechte"
- L2 "Mitarbeiter"
  - L3 "Personaldaten"
  - L3 Neueinstellung
    - L4 Berufseinsteiger
  - L3 "Führungskraft"
- L2 "Betriebsrat"
- L2 "Geschäftsbereiche"
- L2 "Finanzabteilung"
- L2 "Personalabteilung"
  - L3 "Personalentwicklungsabteilung"

- L1 • Definition Personalentwicklung
  - L2 • "Herausforderungen"
  - L2 • Aufgaben der Personalentwicklung
    - L3 • Zielgerichtete Entwicklung des Mitarbeiters
    - L3 • Mitarbeiterkompetenz halten
    - L3 • Unternehmenskompetenz halten
    - L3 • Unternehmenswerte vermitteln
    - L3 • "Mitarbeitermotivation"
    - L3 • "Mitarbeiterbindung"
- L1 • Aktuelle Situation
  - L2 • Unternehmensgröße
  - L2 • Branche
  - L2 • Macht der Personalabteilung
  - L2 • Position des Interviewten
  - L2 • Bestehende Tools
  - L2 • Konzepterstellung
- L1 • "Ausbildung"
- L1 • "Mitarbeiterbefragung"
- L1 • "Verbesserungsmanagement"
- L1 • "Einverständniserklärungen"
- L1 • "Backup während Entwicklungsmaßnahme"
- L1 • " Entwicklungsmaßnahmen testen"
- L1 • "Budgetplanung"
- L1 • "Performance Improvement Plans"

Appendix B - The code system for requirements specification

*L = hierarchical level

L0 • Code System

- L1 • Akteure
  - L2 • Externe Akteure
    - L3 • Externe
    - L3 • Kunde
  - L2 • Interne Akteure
    - L3 • Betriebsrat
    - L3 • Board
    - L3 • Fachabteilung
    - L3 • Mitarbeiter
      - L4 • Mentor
      - L4 • Neuer Mitarbeiter
    - L3 • Personalabteilung
      - L4 • Personaler
    - L3 • Projektleiter
    - L3 • Sekretariat
    - L3 • Teilnehmer
    - L3 • Vorgesetzter
- L1 • Background-Information
  - L3 • Interviewee
    - L4 • Position im Unternehmen
  - L3 • Unternehmensgröße
  - L3 • Zeit des Interviews
- L1 • Personalentwicklung
  - L2 • Aufgabenbereiche
    - L3 • Ausbildung
    - L3 • Auswertungen
      - L4 • Auswertung exportieren
        - L5 • als Excel Datei
      - L4 • Query erstellen
      - L4 • Report erstellen

- L4 • Suchfunktion
- L3 • Dokumentation
  - L4 • Datenstammblatt erfassen
    - L5 • Falls es Änderungen gibt
      - L6 • Änderungen dokumentieren
    - L5 • Falls Mitarbeiter neu ist
      - L6 • Personenbezogene Daten erfassen
        - L7 • Adresse
        - L7 • Geburtsdatum
        - L7 • Geburtsort
        - L7 • Name
    - L5 • Performance Development Systeme
      - L6 • Gehaltsentwicklung
      - L6 • Jobentwicklung
      - L6 • Performance Development
    - L5 • Qualifikationsprofil
      - L6 • Belegte Kurse
      - L6 • CV
      - L6 • Falls Berater
        - L7 • Druckfunktion
      - L6 • Gehalt
      - L6 • Hierarchiestufe
      - L6 • Rolle
    - L5 • Zugriffsrechte erteilen
      - L6 • Art
        - L7 • Eingeschränkte Zugriffsrechte
        - L7 • Uneingeschränkte Zugriffsrechte
      - L6 • Datenschutzrecht beachten
- L3 • Führungskräfteentwicklung
- L3 • Leistungsbeurteilungszyklus
  - L4 • Evaluation der Mitarbeiter
    - L5 • Kompetenzen
      - L6 • Anforderungskatalog
        - L7 • Kriterien
      - L6 • Art

L7 • Fachwissen

L7 • Soziale Kompetenzen

L6 • Kompetenzen beurteilen

L7 • Assessment durchführen

L5 • Mitarbeiterfeedback

L6 • 360 Grad Feedback

L6 • Fragebögen erstellen

L6 • Mitarbeiterfeedback auswerten

L6 • Mitteilung senden

L7 • per Email

L5 • Mitarbeitergespräch

L6 • Beratungsgespräch

L6 • Einstiegsgespräch

L6 • Endjahresgespräch

L7 • am Jahresende

L7 • Beförderungen festlegen

L7 • Leistungsfaktor festlegen

L7 • Peer-to-Peer Vergleich

L6 • Gesprächsergebnisse dokumentieren

L6 • Halbjahresgespräch

L7 • Stand der Mitarbeiter kontrollieren

L7 • zur Mitte des Jahres

L6 • Leistungsbeurteilungsformular initiieren

L7 • Erwartungshaltungen festlegen

L7 • Weiterbildungsbedarf festlegen

L6 • Organisation

L7 • Leitfaden erstellen

L7 • Termin einpflegen

L7 • Termin mitteilen

L8 • via Email

L6 • Projektbeurteilung

L7 • während des Jahres

L6 • Ziele

L7 • Ziele dokumentieren

L7 • Ziele evaluieren

L7 • Ziele festlegen

L8 • Art

L9 • qualitativ

L9 • quantitativ

L9 • strategisch

L3 • Mitarbeiterbefragung

L3 • Mitarbeiterbindung

L4 • Beförderungen

L4 • Gehaltserhöhung

L3 • Onboarding

L4 • Einarbeitungsplan mitteilen

L5 • OneNote

L5 • über Kalender

L5 • via Email

L4 • Falls Mitarbeiter neu ist

L4 • Mentorship

L3 • Weiterbildungsangebot

L4 • Anmeldungsverfahren

L5 • Bewerber meldet sich an

L5 • Falls maximale Anzahl der Bewerber überschritten

L6 • Fachbereich wählt aus

L5 • Falls virtuelle Schulung

L6 • Zielgruppe prüfen

L5 • Vorgesetzter genehmigt

L4 • Anzahl der Teilnehmer erfassen

L4 • Art

L5 • Academies

L6 • Standort

L5 • Coaching

L5 • Virtuell/E-Learning

L4 • Auslöser

L5 • Mitarbeitergespräche

L5 • Neueinstellung

L5 • Rollenwechsel

L4 • Erfolg der Maßnahme messen

- L4 • Feedbackbögen ausfüllen
- L4 • Katalog erstellen
- L4 • Schulung planen
  - L5 • Arbeitsausfall erfassen
  - L5 • Bewirtung planen
  - L5 • Personal Backup planen
  - L5 • Raum buchen
- L4 • Trainingskosten erfassen
  - L5 • Belege speichern
  - L5 • Kostenstelle erfassen
  - L5 • ROI dokumentieren
- L4 • Über Weiterbildung informieren
  - L5 • Über Intranet

L2 • Herausforderung

L2 • Prozessbeschreibung/Beispiel

Appendix C - Requirements Specification

Req. 1: Leistungsbearbeitungsformular

Die HR-Software sollte dem Vorgesetzten und dem Mitarbeiter die Möglichkeit bieten Leistungsbearbeitungsformular zu initiieren.

Verknüpfte Codes:

Anforderungstyp: funktional

Req 1.1: Erwartungshaltungen

Die HR-Software sollte dem Vorgesetzten und dem Mitarbeiter die Möglichkeit bieten Erwartungshaltungen festzulegen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 1.2: Bildungsbedarf

Die HR-Software sollte dem Vorgesetzten und dem Mitarbeiter die Möglichkeit bieten Bildungsbedarf festzulegen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2: Evaluation der Mitarbeiter

Die HR-Software sollte dem Nutzer die Möglichkeit bieten einen Mitarbeiter zu evaluieren.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.1: Mitarbeitergespräche

Die HR-Software sollte dem Personaler und Vorgesetzten die Möglichkeit bieten Mitarbeitergespräche zu organisieren.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.1.1: Gesprächsleitfaden

Die HR-Software sollte dem Personaler die Möglichkeit bieten Gesprächsleitfaden zu erstellen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.1.2: Gesprächsergebnisse

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Gesprächsergebnisse zu dokumentieren.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.1.3: Mitarbeiterstand

Zur Mitte des Jahres sollte die HR-Software dem internen Nutzer die Möglichkeit bieten den Mitarbeiterstand zu kontrollieren.

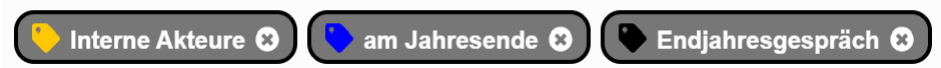
Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.1.4: Endjahresgespräch

Am Jahresende sollte die HR-Software dem internen Nutzer die Möglichkeit bieten Endjahresgespräch durchzuführen.

Verknüpfte Codes:



Anforderungstyp: funktional

Req. 2.1.4.1: Beförderungen

Am Jahresende sollte die HR-Software dem internen Nutzer die Möglichkeit bieten Beförderungen festzulegen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.1.4.2: Leistungsfaktor

Am Jahresende sollte die HR-Software dem internen Nutzer die Möglichkeit bieten Leistungsfaktor festzulegen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.1.4.3: Peer-to-Peer Vergleich

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Peer-to-Peer Vergleich durchzuführen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.1.5: Projektbeurteilungen

Während des Jahres sollte die HR-Software dem Projektleiter die Möglichkeit bieten Projektbeurteilungen zu erstellen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.1.6: Terminplanung

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten den Termin einzupflegen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.1.7: Mitteilung des Termins

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten dem Mitarbeiter den Termin mitzuteilen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.1.8: Zieldokumentation

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Ziele zu dokumentieren.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.1.9: Zielevaluation

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Ziele zu evaluieren.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.2: Kompetenzen

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Kompetenzen zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.2.1: Anforderungskatalog

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Anforderungskatalog zu erstellen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.2.1.1: Kriterien

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Kriterien einzupflegen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.2.2: Kompetenzbeurteilung

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Kompetenzen zu beurteilen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.2.2.1: Assessment

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Assessment durchzuführen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.3: Mitarbeiterfeedback

Die HR-Software sollte dem Nutzer die Möglichkeit bieten Mitarbeiterfeedback zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.3.1: 360 Grad Feedback

Die HR-Software sollte dem Nutzer die Möglichkeit bieten 360 Grad Feedback zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.3.2: Auswertung des Mitarbeiterfeedbacks

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Mitarbeiterfeedback auszuwerten.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.3.3: Fragebögen

Die HR-Software sollte der Personalabteilung und den Externen die Möglichkeit bieten Fragebögen zu erstellen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 2.3.4: Mitteilung

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten eine Mitteilung per Email zu senden.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3: Weiterbildungsangebot

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Weiterbildungsangebot anzubieten.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.1: Anmeldung

Die HR-Software sollte dem Mitarbeiter die Möglichkeit bieten sich als Bewerber anzumelden.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.2: Auswahl der Teilnehmer

Falls die maximale Anzahl der Bewerber überschritten ist, muss die HR-Software dem Fachbereich die Möglichkeit bieten Bewerber auszuwählen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.3: Prüfung der Zielgruppe

Falls es eine virtuelle Schulung ist, muss die HR-Software der Fachabteilung und Personalabteilung die Möglichkeit bieten die Zielgruppe zu prüfen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.4: Teilnehmeranzahl

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten die Anzahl der Teilnehmer zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.5: Genehmigung

Die HR-Software muss dem Vorgesetzten die Möglichkeit bieten die Bewerbung zu genehmigen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.6: Feedbackbogen

Die HR-Software sollte dem Teilnehmer die Möglichkeit bieten Feedbackbogen auszufüllen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.7: Information

Die HR-Software sollte dem Nutzer die Möglichkeit bieten über die Weiterbildungsmaßnahme über Intranet zu informieren.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.8: Planung

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten die Schulung zu planen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.8.1: Arbeitsausfall

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten den Arbeitsausfall zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.8.2: Personal-Backup

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Personal Backup zu planen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.8.3: Bewirtung

Die HR-Software sollte dem Sekretariat die Möglichkeit bieten die Bewirtung zu planen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.8.4: Raumbuchung

Die HR-Software sollte dem Sekretariat die Möglichkeit bieten den Raum zu buchen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.9: Trainingskosten

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten die Trainingskosten zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.9.1: Belege

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten die Belege zu speichern.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.9.2: Kostenstelle

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten die Kostenstelle zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.9.3: ROI

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten ROI zu dokumentieren.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.10: Katalog

Die HR-Software sollte der Personalabteilung die Möglichkeit bieten einen Katalog zu erstellen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 3.11: Erfolgsmessung

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Erfolg der Maßnahme zu messen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 4: Auswertungen

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Auswertung zu erstellen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 4.1: Report

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten einen Report zu erstellen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 4.2: Query

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten eine Query zu erstellen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 4.3: Export

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten eine Auswertung als Excel Datei zu exportieren.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 5: Mitarbeiterbefragung

Die HR-Software sollte der Personalabteilung die Möglichkeit bieten Mitarbeiterbefragung zu organisieren.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 6: Onboarding

Falls ein Mitarbeiter neu ist, sollte die HR-Software dem Personaler die Möglichkeit bieten die Einarbeitung zu planen.

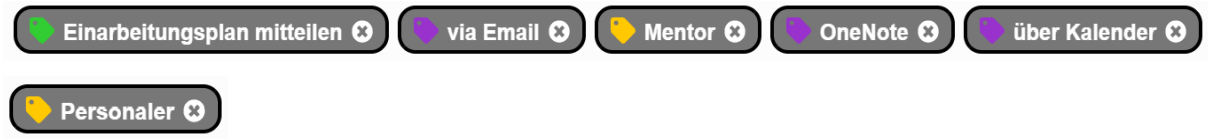
Verknüpfte Codes:

Anforderungstyp: funktional

Req. 6.1: Mitteilung

Die HR-Software sollte dem Personaler und dem Mentor die Möglichkeit bieten den neuen Mitarbeiter über den Einarbeitungsplan via Email, Kalender oder OneNote zu informieren.

Verknüpfte Codes:



Anforderungstyp: funktional

Req. 7: Dokumentation

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Datenstammblatt zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.1: Personenbezogene Daten

Falls ein Mitarbeiter neu ist, sollte die HR-Software dem Nutzer die Möglichkeit bieten personenbezogene Daten zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.1.1: Adresse

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Adresse zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.1.2: Geburtsdatum

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Geburtsdatum zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.1.3: Geburtsort

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Geburtsort zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.1.4: Name

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten den Namen zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.2: Änderungen

Falls es Änderungen gibt, sollte die HR-Software dem internen Nutzer die Möglichkeit bieten Änderungen zu dokumentieren.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.3: Zugriffsrechte

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Zugriffsrechte zu erteilen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.3.1: Datenschutzrecht

Die HR-Software sollte dem Betriebsrat die Möglichkeit bieten das Datenschutzrecht zu beachten.

Verknüpfte Codes:

Anforderungstyp: nicht funktional

Req. 7.4: Performance Development

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Performance Development zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.4.1: Gehaltsentwicklung

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Gehaltsentwicklung zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.4.2: Jobentwicklung

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Jobentwicklung zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.5: Qualifikationsprofil

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Qualifikationsprofil zu erstellen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.5.1: Belegte Kurse

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten die belegten Kurse zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.5.2: CV

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten CV zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.5.3: Gehalt

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Gehalt zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.5.4: Hierarchiestufe

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Hierarchiestufe zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.5.5: Rolle

Die HR-Software sollte dem internen Nutzer die Möglichkeit bieten Rolle zu erfassen.

Verknüpfte Codes:

Anforderungstyp: funktional

Req. 7.5.6: Druckfunktion

Falls es sich um einen Berater handelt, sollte die HR-Software dem internen Nutzer und dem Kunden die Möglichkeit bieten das Qualifikationsprofil auszudrucken.

Verknüpfte Codes:

Anforderungstyp: funktional

References

Barber, J. P., & Walczak, K. K. (2009). Conscience and Critic: Peer Debriefing Strategies in Grounded Theory Research. https://www.researchgate.net/publication/242479874_Conscience_and_Critic_Peer_Debriefing_Strategies_in_Grounded_Theory_Research

Bell, D. (2003). UML basics: An introduction to the Unified Modeling Language.

Bourque, P., & Fairley, R. E. (2014). *SWEBOK Guide V3: Guide to the Software Engineering-Body of Knowledge*. IEEE Computer Society. http://www.swebok.org

Carvalho, L., Scott, L., & Jeffery, R. (2005). An exploratory study into the use of qualitative research methods in descriptive process modelling. *Information and Software Technology*, *47*(2), 113–127. https://doi.org/10.1016/j.infsof.2004.06.005

Chakraborty, S., & Dehlinger, J. (2009). Applying the Grounded Theory Method to Derive Enterprise System Requirements 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, 2009, pp. 333-338,, 333–338. https://doi.org/10.1109/SNPD.2009.102

Chakraborty, S., Rosenkranz, C., & Dehlinger, J. (2015). Getting to the shalls: Facilitating sensemaking in requirements engineering. *ACM Transactions on Management Information Systems*, *5*(3), Article 14, 1–30. https://doi.org/10.1145/2629351

Chung, L., & do Prado Leite, J. C. S. (2009). On Non-Functional Requirements in Software Engineering. In Borgida, Alexander T., Chaudhri, Vinay K., P. Giorgini, & E. S. Yu (Eds.), *Lecture Notes in Computer Science: Vol. 5600. Conceptual Modeling: Foundations and Applications* (Vol. 5600, pp. 363–379). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-02463-4_19

Cleland-Huang, J., Settimi, R., Zou, X., & Solc, P. (2007). Automated classification of non-functional requirements. *Requirements Engineering*, *12*(2), 103–120. https://doi.org/10.1007/s00766-007-0045-1

Corbin, J., & Strauss, A. (1990a). *Basics of qualitative research: Grounded theory procedures and techniques*. Sage Publications.

Corbin, J., & Strauss, A. (1990b). Grounded Theory Research: Procedures, Canons, and Evaluative Criteria. *Qualitative Sociology*(13), 3–21. https://doi.org/10.1007/BF00988593

Escalona, M. J [Maria José], & Koch, N. (2007). Metamodeling the Requirements of Web Systems. In J. Filipe, J. Cordeiro, & V. Pedrosa (Eds.), *Web Information Systems and Technologies: International Conferences WEBIST 2005 and WEBIST 2006, Revised Selected Papers* (1st ed., Vol. 1, pp. 267–280). Springer Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-74063-6_21

Fernández, D. M., Wagner, S., Kalinowski, M., Felderer, M., Mafra, P., Vetrò, A., Conte, T., Christiansson, M.-T., Greer, D., Lassenius, C., Männistö, T., Nayabi, M., Oivo, M., Penzenstadler, B., Pfahl, D., Prikladnicki, R., Ruhe, G., Schekelmann, A., Sen, S., . . . Wieringa, R. (2017). Naming the pain in requirements engineering. *Empirical Software Engineering*, *22*(5), 2298–2338. https://doi.org/10.1007/s10664-016-9451-7

Franch, X. (1998). Systematic formulation of non-functional characteristics of software, 174–181. https://doi.org/10.1109/ICRE.1998.667823

Genero, M., Piattini, M., & Calero, C. (2005). A Survey of Metrics for UML Class Diagrams. *The Journal of Object Technology*, *4*(9), 59. https://doi.org/10.5381/jot.2005.4.9.a1

Glaser, B. G., & Strauss, A. (1967). *The Discovery of Grounded Theory*. Aldine.

Halaweh, M. (2012). Using grounded theory as a method for system requirements analysis.

IEEE Computer Society (1998). IEEE Recommended Practice for Software Requirements Specifications. Advance online publication. https://doi.org/10.1109/IEEESTD.1998.88286

Jeusfeld, M. A. (2009). Metamodel. In LIU, L., ÖZSU, M.T. (Ed.), *Encyclopedia of Database Systems* (pp. 1727–1730). Springer, Boston, MA. https://doi.org/10.1007/978-0-387-39940-9_898

Kassab, M., Ormandjieva, O., & Daneva, M. (2009). A Metamodel for Tracing Non-functional Requirements. *2009 WRI World Congress on Computer Science and Information Engineering*, 687–694. https://doi.org/10.1109/CSIE.2009.946

Kaufmann, A. (2021). *Domain Modeling Using Qualitative Data Analysis* [Dissertation]. Friedrich-Alexander-Universität Erlangen-Nürnberg. https://opus4.kobv.de/opus4-fau/files/16736/AndreasKaufmannDissertation.pdf

Kaufmann, A., Krause, J., Harutyunyan, N., Barcomb, A., & Riehle, D. (2022). A validation of QDAcity-RE for domain modeling using qualitative data analysis. *Requirements Engineering*, *27*(1), 31–51. https://doi.org/10.1007/s00766-021-00360-6

Kaufmann, A., & Riehle, D. (2015). Improving traceability of requirements through qualitative data analysis. In U. Aßmann, B. Demuth, T. Spitta, G. Püschel, & R. Kaiser (Eds.), *Lecture Notes in Informatics (LNI) - Proceedings: P-239. Software-engineering and management 2015* (pp. 165–170). Gesellschaft für Informatik e.V. https://dl.gi.de/bitstream/handle/20.500.12116/2543/165.pdf?sequence=1

Kaufmann, A., & Riehle, D. (2019). The QDAcity-RE method for structural domain modeling using qualitative data analysis. *Requirements Engineering*, *24*(1), 85–102. https://doi.org/10.1007/s00766-017-0284-8

Kluge, R., Schlör, K., & SOPHIST GmbH. (2016). *Requirements-Engineering - A short RE Primer* (1st ed.). Flyeralarm. https://www.sophist.de/fileadmin/user_upload/Bilder_zu_Seiten/Publikationen/Wissen_for_free/RE-Broschuere_Englisch_-_Online.pdf

Kluge, R., & SOPHIST GmbH. (2016). *MASTER – Schablonen für alle Fälle* (3rd ed.). Flyeralarm. https://www.sophist.de/fileadmin/user_upload/Bilder_zu_Seiten/Publikationen/Wissen_for_free/MASTeR_Broschuere_3-Auflage_interaktiv.pdf

Kluge, R., & SOPHIST GmbH. (2021). *Requirements-Engineering - Clever & kompakt: Die RE-Fibel von SOPHIST* (1st ed.). Flyeralarm. https://www.sophist.de/fileadmin/user_upload/Bilder_zu_Seiten/Publikationen/Wissen_for_free/RE-Fibel_Int/RE-Fibel_1-Auflage_Interaktiv_2021.pdf

Kotonya, G., Sommerville, Ian. (1998). *Requirements engineering: processes and techniques*. Wiley.

Kunz, K. (2015). *Developing a Domain Analysis Procedure based on Grounded Theory Method* [Master thesis]. Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany. https://oss.cs.fau.de/wp-content/uploads/2015/06/kunz_2015_arbeit.pdf

Pozgaj, Z., Sertic, H., & Boban, M. (2003). Effective requirement specification as a precondition for successful software development project, 669–674. https://doi.org/10.1109/ITI.2003.1225420

Rastogi, V. (2015). Software Development Life Cycle Models - Comparison, Consequences. *(IJCSIT) International Journal of Computer Science and Information Technologies*, *6*, 168–172.

Salow, S. (2016). *A Metamodel for Code Systems* [Master thesis]. Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany. https://oss.cs.fau.de/wp-content/uploads/2016/10/salow-2016-arbeit.pdf

Schön, E.-M., Sedeño, J., Mejías, M., Thomaschewski, J., & Escalona, M. J [María José] (2019). A Metamodel for Agile Requirements Engineering. *Journal of Computer and Communications*, *07*(02), 1–22. https://doi.org/10.4236/jcc.2019.72001

Shen, W., Compton, K., & Huggins, J. (2002). A toolset for supporting UML static and dynamic model checking, 147–152. https://doi.org/10.1109/CMPSAC.2002.1044545

SOPHIST. *MASTER - Die SOPHIST-Templates für Requirements.* SOPHIST GmbH. https://www.sophist.de/fileadmin/user_upload/Bilder_zu_Seiten/Publikationen/Wissen_for_free/MASTeR_-Kern_RE-Plakat2.pdf

Ya'u, B. I., Nordin, A., & Salleh, N. (2016). Software Requirements Patterns and Meta Model: A Strategy for Enhancing Requirements Reuse (RR), 188–193. https://doi.org/10.1109/ICT4M.2016.047