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

Sindy Salow

A Metamodel for Code Systems

Submitted on 05.10.2016

Supervisor: Prof. Dr. Dirk Riehle; Andreas Kaufmann, M. Sc.
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.

Sindy Salow

Frankfurt am Main, 05.10.2016

License

Sindy Salow

Frankfurt am Main, 05.10.2016

Abstract

Requirements elicitation is an important factor in software engineering. Mainly the information needed is elicited through interviews and other qualitative sources. The analysis that follows is often an ad-hoc process that relies on expertise of the analyst(s) and therefore is hardly replicable. Additionally, the process is not transparent as the resulting modeling elements cannot be mapped to the initial data. First attempts to solve this issues by adapting the clearly defined steps of Qualitative Data Analysis (QDA) suggest that the approach should be followed up. In order to further formalize the process this thesis suggests a metamodel which allows to derive structure and behavior models from the same coding process. The metamodel is derived by analyzing an existing metamodel and by comparing different existing coding systems and their resulting modeling artifacts. The metamodel is extended with a rule system and tested on an exemplary data set. For validation the resulting models are compared to models from an ad-hoc modeling process and evaluated by experts. Results show that utilizing QDA with a code system metamodel allows for an increase in transparency and makes it more easy to vary detail levels of the derived models.

Keywords: Domain Model, Domain Analysis, Requirements Engineering, Qualitative Data Analysis (QDA)

Content

List of Figures

List of Tables

List of Abbreviations

CAQDAS	Computer assisted Qualitative Data Analysis Software
HR	Human Resources
QDA	Qualitative Data Analysis
UML	Unified Modeling Language

1 Introduction

1.1 Original Thesis Goals

Requirements are elicited through interviews and other qualitative sources. The analysis that follows is often an ad-hoc process that relies on expertise of the analyst(s) and therefore is hardly replicable. Additionally, the process is not transparent as the resulting modeling elements cannot be mapped to the initial data. First attempts to solve this issues by adapting the clearly defined steps of Qualitative Data Analysis (QDA) suggest that the approach should be followed up.
In order to further formalize the process this thesis suggests a metamodel which allows to derive structural and behavioral models from the same coding process. The metamodel is derived by analyzing an existing metamodel and by comparing different existing coding constructs and their resulting modeling artifacts. The metamodel is used on an exemplary data set and the resulting models are compared to models from an ad-hoc modeling process. Expert feedback is used to further validate the models.

1.2 Changes to Thesis Goals

The goals of this thesis were not changed.

2 Research Chapter

2.1 Introduction

By utilizing Qualitative Data Analysis (QDA) in combination with a code system metamodel it is possible to create consistent and complete models for requirements engineering in a more structured, detailed and transparent manner.

Requirements engineering starts with elicitation of the necessary information usually in form of qualitative data. Completeness of this information, the quality of the processing and transformation into abstract representations have a high impact on the requirement specifications and therefore on the project success. Evaluations by the Standish Group (2004) show that poor requirements specifications are the leading cause for the high failure rate of software projects. Problems and errors discovered in late phases of projects cause much higher costs than those discovered early during requirements elicitation and analysis (Moody, 2005). Regardless, incomplete and inconsistent requirements are still identified as a major problem in projects (Fernandez & Wagner, 2013). This is caused by a gap between domain experts who provide the information, and system analysts who process it. A system analyst, evaluating the relevant information, has no means to prove the information is complete (Chakraborty & Dehlinger, 2009) and additionally faces ambiguous terms as a challenge during process and domain analysis (Flowers & Edeki, 2013).

A positive effect on the specification of business requirements can be created through process models that allow a consistent and detailed understanding of a domain (Cardoso, Almeida & Guizzardi, 2009). Process models include activities and their causal and temporal relationships, as well as specific business rules (Van der Aalst, Ter Hofstede & Weske, 2003). For modeling and engineering complex systems it is necessary to take their ecosystem into account (Rausch, Bartelt, Herold, Klus, & Niebuhr, 2013) which means to also present structural elements. A domain model combines these demands as it identifies business- and application-specific entity types and relationships (Broy, 2013). Domain models can be represented through different known models from software engineering. In using appropriate behavioral and structural models companies can leverage further advantages. Structural models support various stages of the software development process, depicting the structure or state of a system (Seidl, Scholz, Huemer & Kappel, 2015). Process models foster process optimization (De Oca, Snoeck, Reijers & Rodríguez-Morffi, 2014) and enable companies to improve their projects as well as overall competitiveness (Geambasu, 2012).

Despite this knowledge, effective and comprehensible methods for the modeling processes are missing (Becker, 2011). More comprehensive de Oca et al. (2014) recently stated that the elicitation, modeling and validation of processes still needs to be addressed by research. Several researchers take into account that software engineering is a highly social activity (Coleman & O’Connor, 2007) and as a result use qualitative methods from the social science to describe, explore or understand their respective information system topic. Qualitative methods emphasize personal experiences and interpretation. They are more concerned with understanding the meaning of social phenomena and focus on links among a larger number of attributes across relatively few cases. (Coleman & O’Connor 2007). For these reasons we further analyze qualitative research methods concerning their suitability for requirements engineering.

Qualitative research methods include grounded theory, which uses a combination of data analysis and data collection. The analytical steps are constant comparison and theoretical sampling which means that data elements are examined and compared and if gaps are identified new data is sampled. Constant comparison as well as other qualitative analysis techniques rely on “coding” the data. Labels (codes) are generated while studying the data in order to identify and describe concepts and features. This is known as open coding (Charmaz, 1996; Corbin &

Strauss, 1990). Further steps would be axial and selective coding – an advancement of categories and identification of core categories followed by memo writing where upcoming ideas and connections are noted (Charmaz, 1996; Corbin & Strauss, 1990). To develop an explanatory framework, comparative analysis is used (Starks & Trinidad, 2007).

This work describes how to approach and transform qualitative data for requirements engineering, giving guidelines how to use the different coding steps and methods in combination with a metamodel. The metamodel, as well as the guidelines for the process, make it possible to neutralize subjectivity of the analyst.

We base our work on a previously suggested metamodel by Kunz (2015) taking into account several insights from explorative work by Schmitt (2015, 2016), Milisterfer (2016) and Salow (2016). The metamodel classifies the codes in order to structure the transmission into structural and behavioral models. This work extends the previous work by testing the metamodel with various datasets and in this process verify and define the elements of the metamodel. Procedural steps of the QDA are tracked, so that we can suggest a thorough method for coding and transforming qualitative data into several models needed for requirements specification. It is highlighted in which manner transparency and replicability are improved and which adaptions were made to the steps of the QDA.

Our contribution is to show that one metamodel for code systems can encompass information needed to create structural and behavioral models and that these models have advancements over ad-hoc models.

In the context of this work the term "code system" describes

- the set of codes and codings (assigned text) themselves,
- the structure (hierarchy of the codes),
- the memos and, where applicable, the color scheme

belonging to one data set and produced through QDA.

To develop this metamodel, we considered various work on QDA which we present in chapter 2.2. The research questions, approach and included data sources are explained independently in the following chapters. The final metamodel, the associated rule system and insights on the procedural steps of the QDA are presented as results in chapter 2.6. Subsequently, chapter 2.7 discusses these results highlighting that the QDA and metamodel based procedure gives the analyst/modeler the chance to produce consistent models, always connect the model elements to the original input and easily increase detail levels when needed. Implications and limitations, suggestions for further research and a short conclusion complete our work.

2.2 Related Work

Even though QDA is used in management studies since the 1970's to analyze natural language and communication patterns (Locke, 2001) its application in requirements engineering is not covered comprehensively by the existing body of research. However, papers by Binder and Edwards (2010), Carvalho, Scott and Jeffery (2005) and Chakraborty and Dehlinger (2009), commonly suggest that there are benefits in using QDA for process modeling and requirements engineering. Findings include the possibility to formalize the modeling process and make it less dependent on the domain knowledge and expertise of the analyst. More specific studies found that by applying concepts from grounded theory approaches, the models are built more systematically (Fernandez & Wagner, 2013), traceability increases (Chakraborty & Dehlinger, 2009; Kaufmann & Riehle, 2015) and especially contextual requirements are more easily identified (Fernandez & Lehmann, 2011; Flowers & Edeki, 2013).

Studies focusing on grounded theory as one of the qualitative research methods include the work by Chakraborty and Dehlinger (2009). They name enterprises a large social system whose processes often lack objective and traceable analysis. In their work they used grounded theory including the systematic procedure of the paradigm model by Strauss and Corbin (1990) to better align enterprise architecture with systems architecture and found that the transition between both is facilitated by the approach. In the conclusion they point out that further research should include investigations on how grounded theory method can support modeling and identify gaps in requirements engineering.

Binder and Edwards (2010), as well as Fernandez and Lehman (2011), further support the conclusion that grounded theory is an eligible method to study the connections between humans and objects in a business environment including the interactions with information and communication technologies and to acquire knowledge about the underlying processes. Small adjustments to acknowledge the research domain and goal, however, seem sensible. This work shows further reasoning concerning the alteration of process steps from grounded theory. Among others the study from Fernandez and Lehman (2011) is taken into account. It implies to preserve the principles of grounded theory but to use different abstraction levels and therefore extend the process of analysis. A modification we did not consider due to high risks of bias is the alteration of the first step of coding by introducing key template categories. It was suggested by Binder and Edwards (2010) along with more applicable guidelines how to execute the different coding steps in order to arrive at a qualitative theory and consequently show how the grounded theory method is applicable to identify business strategies.

One comprehensive work by Chakraborty, Rosenkranz and Dehlinger (2015) not only linked grounded theory to information systems and process optimization but also to requirements engineering. They developed a systematic and traceable procedure for non-functional requirements. The Grounded and Linguistic-Based Requirements Analysis Procedure (GLAP) was designed to support sensemaking in requirements engineering. Grounded theory method was included in this procedure to ensure a systematic procedure as well as traceability and the ability to operationalize. We build on their proposition that grounded theory is able to bridge the gap between qualitative data input and formal output. This is supported by a study of Panayiotou, Gayialis, Evangelopoulos and Katimertzoglou (2015) which showed that a structured requirements engineering framework can improve requirements engineering and the transition into the final system.

Still the quality of the resulting models needs to be assessed. Our results of the QDA-based approach will be compared to an ad-hoc model as Carvalho et al. (2005) found important differences between an ad-hoc model of an experienced analyst and one created with the help of constant comparison. Among other things they differed concerning data coverage and sensible representation of hierarchies. Similarly, Milisterfer (2016) compared a domain model created ad-hoc with a previously by Kunz (2015) created domain model based on QDA and findings suggested that the grounded theory based methodology by Kunz identified additional concepts and more structural relations. The methodology of Kunz (2015) introduced a metamodel for the structuring of the coded data before the transformation. This concept cannot be found in other studies although a metamodel allows more structure and standardization in the transformation process and standardization concerning modeling approaches is critical area of business process modeling (Indulska, Recker, Rosemann & Green, 2009). Relating to process models, metamodels are used to define parameters of UML models (OMG, 2005) or to evaluate them regarding certain qualities (List & Korherr, 2006).

We see the metamodel by Kunz (2015) as a good foundation for a structured and integrated approach especially as it is suitable for structural and behavioral models. This is of importance to work on requirements because a holistic view on processes needs to be supplemented by other views to represent all the real-world constructs required (e.g. Green & Rosemann, 2000).

2.3 Research Question

1. Can one metamodel for code systems encompass information needed to create structural and behavioral models?

The first research question asks if

- the elements of the metamodel are all used,
- are sensible connected within one model
- and if the structuring and coding of qualitative data based on the metamodel allow the creation of structural and behavioral models.

2. Do the models created with the metamodel have advancements in terms of consistency, completeness, transparency and replicability compared to ad-hoc models?

For the second questions the following characteristics will be examined:

- Consistency: There is no contradiction between the parts/elements of one model and among the different derived models.
- Completeness: All facts and ideas represented in the input data are represented in the models.
- Transparency: The connection between input data and resulting model can be traced in every phase.
- Replicability: The resulting model would not vary if another person or the same person at another time repeats the modeling process.

2.4 Research Approach

The overarching approach is leaned on principles of action research where the attempt to solve an existing problem is accompanied by studying the experience of solving the problem (Davison, Martinsons & Kock, 2004). We applied action research as an iterative approach where new information occurring through the process is integrated into the theory (Easterbrook, Singer, Storey & Damian, 2008). Therefore, we conducted several analysis and test steps to adapt and extend the metamodel. Furthermore, we reflected on the explorative process of analyzing and utilizing the metamodel and adjusted our method accordingly. The different steps are explained in the following sub-chapters. All details on the used data sources are separately described in chapter 2.5.

The metamodel is developed to allow a transfer of data gained from QDA into structural and behavioral models and to provide orientation during the QDA in itself. Prompted by previous work (see chapter 2.2) we adopted mostly methods from the grounded theory, like open and axial coding. The single process steps are explained as part of the resulting procedure in chapter 2.6.3.

The processing of the data was executed with MaxQDA, a Computer assisted Qualitative Data Analysis Software (CAQDAS). Information concerning the codes can be inserted in the code memos, but no features for code metadata or to place additional structured information are currently available. To ensure transparency during analysis we chose to execute parts of the labeling in Microsoft Excel. For the graphical representation of domain models we use UML class models for the structural part and UML activity diagrams for the behavioral part. Therefore, entities identified during the coding process are, for example, represented as class in the class model and as object node in the activity diagram (Daoust, 2012).

2.4.1 Adaption of metamodel

First the metamodel and research work by Kunz (2015) was examined in regard to integrity and applicability to structural and behavioral models. We analyzed the labels from the code memos as well as the code structure in combination with the class model Kunz (2015) created to represent the domain. Additionally, we explored if it was possible to create an activity diagram based on Kunz (2015) code system. To do so we added additional labels to the memos conform to the initial metamodel. Two activity diagrams from Salow (2016) extracted from the same data were used to compare complexity and coverage.

Based on the insights from this analysis the metamodel structure and relationship elements were changed. The codes and labels from Kunz were then re-examined in order to ensure that the changes did not hinder or eliminate transitions from code system to models that was possible before.

2.4.2 Rules development and test

The now existing second version of the metamodel was evaluated by applying it to a code system developed by Salow (2016). The code system was developed without a metamodel but on the same data set as used by Kunz (2015). The parallelism in data made it possible to compare the code and label structure as well as the resulting models. During the process of adding labels to the code system by Salow (2016) further adaptions of the metamodel were conducted and a rule system for applying the labels was implemented. These iterative changes and additions were resolutions of occurring problems which are described in detail in chapter 2.6. This process of labeling, analyzing, modeling and correcting was then repeated with two more data sets from other domains to ensure general validity of the findings. Domains covered up to this point were Human Resources (HR) (data from Kunz, 2015; Salow, 2016), Qualitative Research in Social Sciences (data from Schmitt, 2016) and the European Train Control System (data from Schmitt, 2015).

2.4.3 Validation

In the previous steps we relied on existing code systems. In order to investigate if the metamodel has a positive effect on the coding steps we now underwent the whole process of domain analysis. The coding steps including the labeling according to the metamodel are described in detail in chapter 2.6.3. Afterwards we used the code structure and labels from the memos to create a class model and two exemplary activity diagrams of different detail level, noting if the transformation process changed in comparison to the previous examples. Action research like this is often criticized as being ad-hoc and subjective (Easterbrook et al., 2008) and in this case subjectivity was increased as all steps were conducted by the same person. Therefore, external input was necessary to create intersubjectivity.

To collect direct feedback on the domain model we conducted two structured expert interviews with members of the Open Source Research Group. They were chosen based on their research focus and interviewed independently to increase reliability (Dorussen, Lenz & Blavoukos, 2005). According to McGraw (1989) as well as Agarwal and Tanniru (1990) structured interviews have advantages concerning the extraction of specific, domain oriented information. We therefore structured the interviews with item statements on perceived semantic quality and user information satisfaction from Poels, Maes, Gailly & Paemeleire (2005). With this we gather insights on how the information conveyed by the model and the domain that is modeled correspond and how the user perceives the information presented (Poels et al., 2005). The underlying quality properties seemed appropriate as they align with other quality measures. For example, with a framework for evaluating UML Schemas by Cherfi, Akoka & Comyn-Wattiau (2002), quality criteria named by Schuette & Rotthowe (1998) and error categories of models identified by Leung & Bolloju (2005).

Both experts evaluated the structural model from the QDA-based procedure as well as one ad-hoc model (structural domain model) for comparison without knowing the source of the models to minimize bias. The ad-hoc model was created by an IT Consultant with familiarity but no extensive former knowledge on Inner Source. He was presented the same data which was used for the coding process but was free to choose which sources were relevant for him.

All models and the interview structure are presented in the appendix.

2.5 Used Data Sources

We used input from four different domains and two fields, social and technical, to ensure that our metamodel supports all kinds of requirements elicitation processes. The specific works were chosen as they already had a sensible data basis, an existing code system from QDA and parts of domain models for comparison. In the following we explain the data basis and the output of previous work as it builds the foundation for and influenced our analysis process. We also adopted some procedures and methods, which we describe in detail in chapter 2.6.3.

2.5.1 Metamodel

Kunz (2015) adapted grounded theory for its utilization in requirements engineering and derived a systematic procedure for domain analysis including a metamodel (Figure 1). The metamodel set the basis for the coding procedure of the QDA and for deriving a domain model including structural and dynamic aspects. Coding in alignment with the metamodel ensured that one code always only encompasses one aspect and is phrased accordingly, e.g. verbs for activities. The relationship type *affects* from the metamodel included the three specifications: performs, is directed at and influences. We adopted the metamodel at the beginning of our analysis with the alteration that *affects* was exchanged for its subgroups for better overview.

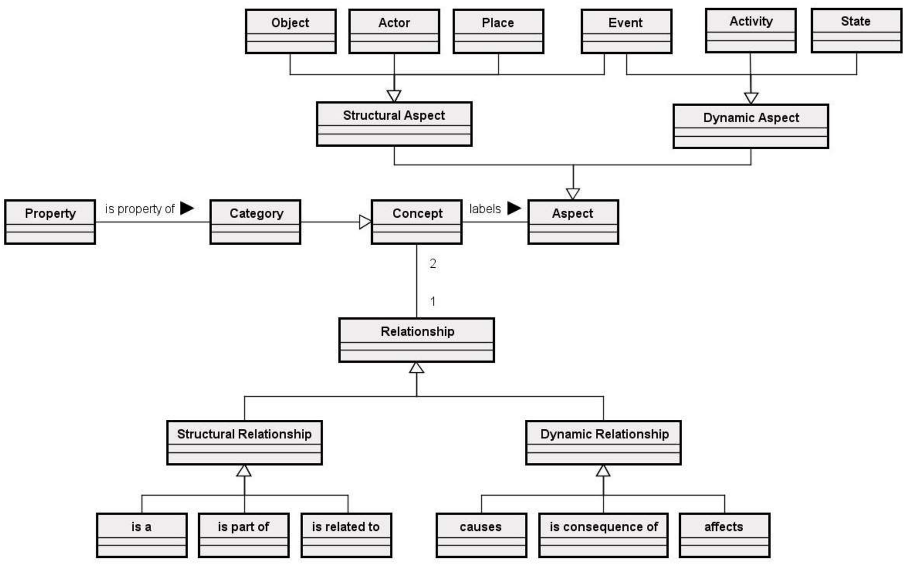


Figure 1: "Code System Meta Model" by Kunz (2015)

2.5.2 Qualitative data input

For the analysis of the first four code systems (Kunz, 2015; Salow, 2016; Schmitt, 2015; Schmitt 2016) we needed to access the initial data input for clarification of context.

Basis for the code system of Kunz (2015) and Salow (2016) are six semi-structured interviews on HR development with four domain experts. The interviews were conducted by Kunz who used additional literature to clarify HR specific terms before transferring the code system into a domain model.

The data on the openETCS was elicitated through interviews as well. After an informal introduction to the project Schmitt (2015) interviewed four team members of the openETCS development team.

For his work on qualitative research Schmitt (2016) carried out five interviews. In addition to insights on QDA, specifications for the QDAcity project were created. Interview partners were four professional researchers from social sciences, who were considered stakeholders in the QDAcity project and the lead developer of QDAcity.

The data set on Inner Source consists mainly of journal and conference papers as well as various publications from companies. One expert interview on Inner Source implementation published as part of a book, two other supporting book chapters and university research complemented the input. University work encompassed six interviews on the implementation of Inner Source at Siemens, excerpts from dissertations and teaching slide decks.

2.5.3 Code systems

The code system developed by Kunz (2015) already is aligned with the metamodel. All information needed for a glossary and for the development of a structural domain model is noted down in the code memos. Not all codes have memos and some memos need complementation to fit the behavioral part of the metamodel.

The code system by Salow (2016) was initially developed without a metamodel, but on the same data set as used by Kunz (2015). Previously, the code hierarchy and a basic color scheme were used to derive activity diagrams from the code system, but no labels or other indicators exist.

Schmitt (2015, 2016) used the memos in their original purpose for the recording of thoughts, further questions, ideas and gaps. Additionally, he added a XML structure in some code memos which contain a glossary text, attributes and relations. Whilst the openETCS project (Schmitt, 2015) was object-orientated, the QDAcity/Social Sciences project (Schmitt, 2016) also included transitions of activities as a behavioral component.

2.5.4 Models

We consulted the "complete domain model" by Kunz (2015) and the activity diagrams by Salow (2016) for reference during the analysis and for comparison with the models build during the analysis of the associated code systems (on HR development).

Similarly, the "Domain Model of the RE tool chain of openETCS" (Schmitt, 2015) and the "Domain Model of the Social Science Domain" (Schmitt, 2016) were used for reference.

2.6 Research Results

This chapter first explains how we arrived at the final metamodel and the corresponding rule-system. We extended the metamodel with two sets of rules, application rules and mapping rules, which both influenced the procedure of the QDA and the subsequent transfer of the codes into a domain model. The description of the procedure explains the single QDA steps as well as the process of labeling and modeling.

At the beginning of our analysis the need for clearly distinguishable terms became immediately apparent as there were several overlaps and uncertainties (e.g. code, label). Table 1 lists the vocabulary essential to comprehend the results including the umbrella terms to distinguish the different metamodel elements.

Term	Explanation
Code	A code is the name assigned to a phenomenon from the original data input. A code can be assigned to several sections of the original data where the phenomenon is represented.
Coding	A coding is one section of the original data input assigned with a code.
labeling; to label	Labeling or to label in this work means to add meta-information to a code (by writing it in the memo). This needs to be done in alignment with the metamodel. (Note: **Label** is an independent noun/term in this work.)
Element	An element refers to one specific part of the metamodel. All elements are listed in table 2. The three types of elements are **Label**, **Aspect** and **Relationship**.
Label	Stands for the elements of the metamodel that are assigned to the code/aspect first (*Concept, Category, Property*) • Every code has to be assigned one **Label** • One code can get only one **Label** • **Label** is used as a flag in memos e.g. **Label** = *Category*
Aspect	Stands for the elements of the metamodel that differentiate the aspect types (*Activity, Process, Object, Place, Actor*) • Comprises structural and dynamic aspects • One code can get only one **Aspect** • **Aspect** is used as a flag in memos e.g. **Aspect** = *Object*
Relationship	Stands for the label that defines dynamic and structural relationships • The code concerned is always the starting point/origin and therefore does not need to be named e.g. *is a*: employee in the code memo from manager means “manager is an employee” • Every relationship element is its own flag in memos e.g. *is a*: employee

Table 1: Terms

Definition of the particular elements of the metamodel are presented in table 2. In the same manner as the other artifacts those definitions were refined throughout the process. For example, *Event* and *is directed at* became obsolete during the analysis and were removed. The definitions of *Process* and *influences* were altered accordingly.

2.6.1 Metamodel

The essential part to label an Aspect through a *Concept*, or its specialization - a *Category*, was already introduced by Kunz (2015) and posed no problem during any analysis step. However, *Property* represents a very different kind of Aspect and therefore was shifted in the metamodel to emphasize this. *Property* is used to give very detailed information, when only a high-level view is needed Aspects labeled with *Property* can be left out.

While analyzing the metamodel elements used by Kunz (2015) we realized that the element *State* was not used once. According to Kunz (2015) it should be used to label something that influences a *Concept* like, for example, a market situation but we could not find codes to apply this element and therefore eliminated it from the metamodel.

The element *Event* was used and at first seemed a very reasonable choice to describe Aspects that represented a structural object and at the same time had a dynamic component. One example was “Project evaluation”; several *Activities* contributed to it, but in the end it was an object/class with certain attributes.

	Element	Definition
Label	*Category*	A *Category* comprises at least one important *Concept*. It describes a phenomenon, idea, process or entity essential to the domain.
	Concept	A *Concept* is a complex phenomenon, idea or attribute dependent on a category or another *Concept*.
	Property	A *Property* is something that contributes to a *Concept* or *Category* without being such itself.
Aspect	*Object*	An **Aspect**, that is a thing/object or a software that can be used (not acts automated/on its own), or the representation of an idea/phenomenon.
	Place	An **Aspect**, that describes a locality – specific or unspecific (e.g. a video conference room, room 302).
	Actor	An **Aspect**, that is a person, a department, a company or an automated agent (e.g. HR, ethic commission, some Trigger).
	~~*Event*~~	~~An Aspect, that is a clearly delimited event which is described as one entity (can be reoccurring)~~.
	Activity	Smallest dynamic entity as defined preliminary to the analysis (depending on the decision upon the level of detail).
	Process	Consists of at least one *Activity*. An **Aspect**, that is a clearly delimited event which is described as one entity (can be reoccurring) (e.g. Training).
Relationship	*is a*	If one **Aspect** extends another **Aspect** (inherits from it). Connects two **Aspects** of the same type. Properties (Attributes/Methods) are inherited.
	is part of	Strong connection; **Aspects** cannot exist without each other.
	is related to	Shows a relation between two **Aspects**. Can be used to map a structural *Property* to a *Concept/Category*.
	is consequence of	Names predecessor of an *Activity*.
	Causes	Names successor of an *Activity*.
	Performs	Describes which activities are carried out by an *Actor*. Always has the form: *Actor* performs *Activity/Process*.
	~~*is directed at*~~	~~Activity or Process changes an Object, Place or Actor.~~
	Influences	*Activity* or *Process* interacts with/changes another **Aspect**. Can be used to map a dynamic *Property* to a *Concept/Category.*

Table 2: Definitions of the metamodel elements

However, as we tried to use it to derive activity diagrams and later when trying to apply it to the code structure by Salow (2016) several discrepancies showed. For example, “Feedback” was labeled as *Category-Event* by Kunz and as such could not be an activity in an activity diagram. The repetitive structure was also not represented adequately. If activities are assigned to an *Event* and given a certain order it needs to be assumed that this order is always kept or it would not be sensible to represent it in a behavioral model. The term *Event* suggests a onetime occurrence. We therefore decided to eliminate the element *Event* and instead use *Process*. Processes can be broken down in sub-processes (*Process* X *is part of Process* Y) until they reach the smallest entity – an *Activity*. This allocation was easy to apply to the following code structures and *Event* as element was not missed. Especially for the code wordings chosen by Schmitt

(2015, 2016) e.g. “mapping from requirements into GitHub items”, “examination of research area” *Process* was a fitting choice and *Event* would not have been appropriate.

In order to establish a sequence for *Process* and *Activity* we maintained *causes* and *is consequence of*. This concept is highly useful for the behavioral models. In all code systems these two Relationships could not be applied thoroughly because highly detailed information on single process steps needs to be available and could not be added to the data samples in the scope of this work. Salow (2016) and Schmitt (2016) focused on behavioral aspects in their code system and therefore more processes and activities could be identified and put in sequence. However, it became apparent that exact wording and adding the sequence directly whilst coding would improve the behavioral models. During the coding of the Inner Source data this was considered but proofed difficult due to the data basis. Most clear sequences could be discovered in the interviews.

As mentioned previously *affects* was split into its subgroups *performs*, *is directed at* and *influences* in the beginning. As *State* was eliminated as element from the very beginning the application scenarios suggested by Kunz (2015) for these terms were changed. At first we used *influences* to connect *Activity/Process* with another dynamic Aspect and *is directed at* for connecting *Activity/Process* with another structural Aspect. This was hard to apply because in order to choose the right Relationship one had to consider both involved codes instead of starting from the current code and knowing which Relationships could be applied. During modeling the HR domain model based on the code system of Salow (2016) and its labels we realized that *is directed at* added no insights or helped the modeling process in any way. It was therefore eliminated and the scenarios for *performs* and *influences* became more clear. We also checked backwards if the substitution of *affects* through its subgroups and the elimination had negative impact on the previous code systems/models and found none. Examples for the application are “Scrum master *performs* Grooming sessions” and “change data *influences* employee data”.

The two structural Relationships *is a* and *is part of* were straight forward in their application and only the rules, how to combine them with *Category/Concept/Property* were adapted. This will be explained in the following chapter.

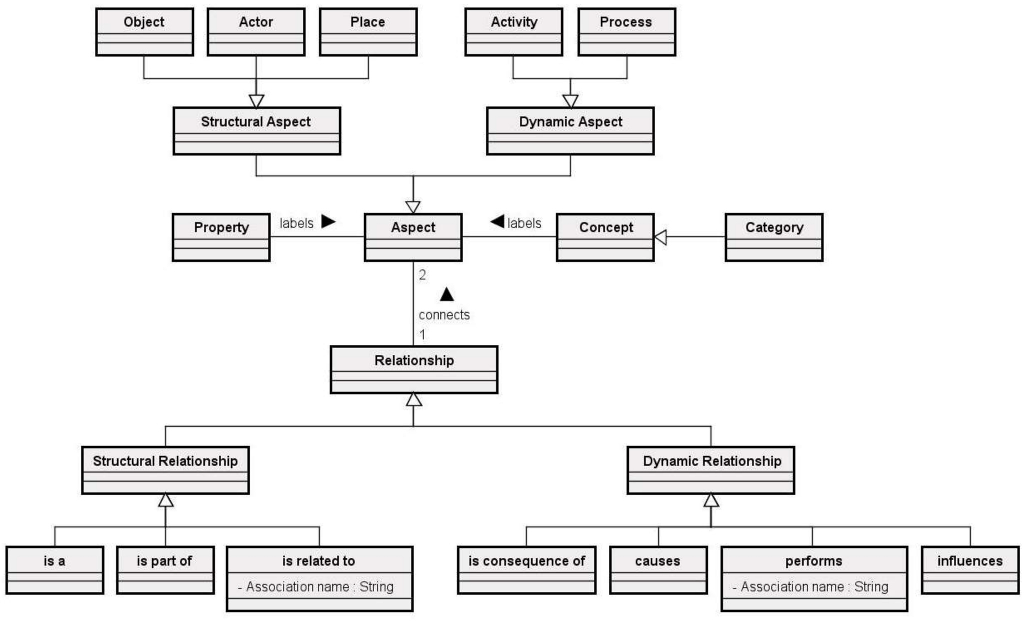


Figure 2: Metamodel

As is visible in figure 2 the final metamodel also includes the attribute “Association name” for *is related to* and *performs.* Both are transferred into associations in structural models (see mapping rules) and were already equipped with association names by Kunz (2015). First, during analyzing the code system by Salow (2016), we only adapted the application method of stating the association names in brackets (Kunz, 2015) where plausible. However, both code systems by Schmitt (2015, 2016) consistently assigned association names which made it much easier to make sense of the connections. As analysts should keep in mind that other stakeholders need this additional insight “Association name” was integrated into the metamodel.

2.6.2 Rules

2.6.2.1 Application rules

The application rules should ensure that the codes are combined in a sensible way. The coder should consider them during the coding/memo-writing process of the QDA. If tool support was available, the software should ensure only the given combinations are used and all mandatory fields are filled immediately.

The rules root from the Label-elements first and then from the Aspect-elements. If the consequence is no strict rule it is not named e.g. Label = Concept: *is related to*-Relationship is optional. During “Rules development and test” the rules were changed several times to take changes made to the metamodel (e.g. eliminate *is directed at)* into account.

Other rules needed some evolution to simplify the process of labeling. For example, we introduced warnings for wrong Label-Aspect combinations (Aspect cannot be *Activity),* missing sequence for *Activity/Process* and missing assignment of *is a/is part of* for *Concept.*

After we came across values for properties several times (e.g. *Concept* “Feedback” had the *Property* “Type” and the values could be “one to one” or “360 degrees”) we decided to allow *is a* for *Properties* in order to flag that this code is a characteristic of this already named *Property. Is a* was chosen because values of an attribute are a specialization of this attribute in the same way classes can be a specialization of other classes (“one to one Feedback” *is a* “Feedback type”).

During the analysis these rules were transferred into an Excel table, to simplify their application and the subsequent mapping. Table 4 is an exemplary section of such a table color coded to show the transfer to the mapping rules.

Condition	Consequence
Label = = (*Category* OR *Concept)*	• **Aspect** cannot be *Activity*
Label = = (*Category* OR *Property)*	• no *is part of* allowed
Label = = *Concept*	• at least one of (*is a, is part of*)
Aspect = = (*Activity* OR *Process*)	• at least one of (*is consequence of*, *causes*)
Aspect = = (*Object* OR *Actor* OR *Place*)	• no *is consequence* allowed • no *causes* allowed • no *influences* allowed
Aspect = = *Actor*	• mandatory *performs*
Aspect ≠ *Actor*	• no *performs* allowed
Label = = *Property* AND **Aspect** = = (*Object* OR *Actor* OR *Place*)	• mandatory *is related to*
Label = = *Property* AND **Aspect** = = (*Activity* OR *Process*)	• mandatory *influences* • no *is related to* allowed

Table 3: Application rules

Code	**Label**	**Aspect**	**Relationship**		
			is a	*is part of*	*is related to*
Distributed Development	*Category*	*Object*			
Open Source	*Category*	*Object*	Distributed Development		Volunteering (depends on)
Volunteering	*Category*	*Object*			Recruiting; Incentive system
OSS development practices	*Concept*	*Object*		Open Source	
OS license	*Property*	*Object*			Open Source
GPL	*Property*	*Object*	OS license		

Table 4: Examples for codes with metamodel information

2.6.2.2 Mapping rules

The mapping rules were roughly outlined as we modeled the structural and behavioral model for the HR development domain and refined as soon as the application rules were finalized. One important improvement adapted from Schmitt (2015, 2016) was to use the flags "START" in *is consequence of* and "END" in *causes* to create a mapping condition for process start and end points.

We applied the mapping rules to create the Inner Source domain model, but had to prioritize and exclude some codes to obtain an uncluttered domain model. Prioritizing and gaps in sequences were the main obstacles in creating the models. Similar to the application rules, the mapping rules would be easier to use/apply if integrated in a CAQDAS or a tool for UML modeling.

Metamodel information of the code	Structural domain model

IF **Label** = = (*Category* OR *Concept*)	CREATE class
	{classname = "code"}

IF **Label** = = (*Category* OR *Concept*)
AND *is a* IS NOT EMPTY
	CREATE generalization
	{origin WHERE classname = = "code";
	target WHERE classname = = "is a"}

IF **Label** = = (*Category* OR *Concept*)
AND *is part of* IS NOT EMPTY
	CREATE aggregation
	{origin WHERE classname = = "code";
	target WHERE classname = = "is part of"}

IF **Label** = = (*Category* OR *Concept*)
AND *is related to* IS NOT EMPTY
	FOR EACH {CREATE association
	{origin WHERE classname = = "code";
	target WHERE classname = = "is related to";
	IF "(name)" EXISTS IN *is related to*
	{associationname = "name"}}}

IF **Label** = = *Property*
AND **Aspect** = = (*Object* OR *Actor* OR *Place*)
AND *is related to* IS NOT EMPTY
	CREATE attribute
	{target WHERE classname = = "is related to";
	propertyname = "code"}

IF **Label** = = *Property*
AND *is a* IS NOT EMPTY
	CREATE value FOR attribute
	{target WHERE classname = = "is a";
	valuename = "code"}

Table 5: Mapping rules (extract)

Table 5 shows an extract from the mapping rules for structural domain models corresponding to the entries in table 4. This overview gives an impression of how the information belonging to the codes could be structured in a CAQDAS and which rules would be needed to automatically map the information into a model. The complete set of mapping rules is presented in appendix E.

2.6.3 Procedure

We consider all activities, performed to arrive at a sufficient set of models to describe the domain, part of the procedure. The first step therefore is to clarify what a sufficient result of the QDA is. QDA serves the description, interpretation and understanding of behavior. We suggest that in a practical case the stakeholders need to agree which behavior the analyst should focus on and which detail level needs to be provided in the domain model. Ideally this is revised after the first iterations of the analysis. As such specification was not given, and time constraints for the analysis existed, we chose a low detail level for the Inner Source domain.

The specific process steps of the QDA we suggest are adapted from the grounded theory method. This systematic set of procedures was initially developed by Glaser and Strauss (1967), and since then adapted and adopted by various researchers. Commonly agreed is that data is iteratively collected, coded and processed and that each iteration gives indications on how to proceed (e.g. Corbin & Strauss, 1990). Although we could not conduct an iterative data collection we recommend to do so. Choosing new data sources to complement one's information (theoretical sampling) is crucial to ensure a complete domain model (theoretical saturation).

Besides the iterative data elicitation, coding and memo writing are the key concepts of the grounded theory method. We adopted the following steps from Kunz (2015), Salow (2016) and Schmitt (2015, 2016) because they had a high overlap, aligned with the theoretical basis of the grounded theory method (e.g. Corbin & Strauss, 1990) and none of the authors stated problems with using them for the development of domain models. To support the application of the metamodel and its rules some additional considerations need to be taken into account:

> **Open coding**: Every idea or phenomenon corresponding with the particular domain is assigned a code. The analyst should ensure that a code only comprises one idea/phenomenon.
>
> **Axial coding**: The previously assigned codes are ordered and grouped. Following the coding paradigm by Corbin and Strauss (1990) the analyst should focus on representing the metamodel relationships in the code hierarchy. Where possible association names should already be added to the memos.
>
> **Selective coding**: Main categories of the domain should be identified and labeled as such. All concepts should be connected to these main categories or need to be re-examined if they actually contribute to the domain. Superfluous concepts or codes without connection to the main categories can be moved outside the code structure used for modeling.
>
> **Memo writing**: Memos comprise notes of the analyst and due to lack of alternatives the labels of the metamodel. We named all metamodel elements in the memo and added labels to them where fitting. This structure can be varied depending on the subsequent modeling process (e.g. XML-structure by Schmitt (2015, 2016)). The analyst should ensure that all application rules are fulfilled while adding the labels.

After assigning ID's the complete code system can be transferred into models as needed. If a process is to complex or a structural model to cluttered to be helpful the code hierarchy can be used to partition processes in sub-processes or respectively to create packages and display the details in separate models.

2.6.4 Validation

We tested our metamodel elements and the rule system by applying it to a data collection on Inner Source. We used all elements from the metamodel except *Place* and we could apply the rule sets without encountering problems.

The resulting behavioral models only showed separate processes but the "Category-Processes" of the domain (e.g. Inner Source Implementation) could not be represented. Firstly, this was a data problem. We could not confirm unclear process steps or fill gaps (no theoretical sampling) and the research papers in our data set did not explain single process steps and their sequence. Secondly, it is a problem of the domain. Open Source and Inner Source processes are described as agile, parallel and distributed, shaped by self-selection, individual decisions, asynchronous and remote communication and tailored to the organization.

The structural model was evaluated by two experts with one ad-hoc model as reference. Different quality criteria were ranked on a Likert scale but the results were inconclusive. Although, between both experts the two models were ranked nearly identical the qualitative assessment highlighted some differences. Inconsistencies and gaps in the ad-hoc model were a given fact but missing elements in our model could be found in the code system and could have easily been added (e.g. "challenges of Open Source"). Another difference between the models was in the focus. The ad-hoc model focuses on technical aspects including elements highlighted as important by expert 2 like software components and tools. This led us to the assumption that the coding steps should be performed by different stakeholders to mitigate bias.

Overall the experts agreed that our model presented the domain mostly correctly, only few elements must be added and no contradictions were present. They stated that the model had advancements over a textual description and would help explain the domain to others.

2.7 Results Discussion

We developed a metamodel and in our validation showed that all elements are applicable and can be combined in a sensible way. A structural and a behavioral model could be derived which affirms our first research question.

When it comes to the advancements proposed in the second research question a more differentiated answer emerges. It is easily answered only for transparency which is better than in ad-hoc models as the connection to the original input is guaranteed via ID's.

The experts did not identify contradictions and therefore we can also state that the structural domain model derived with our method is consistent. However, the same is true for the opposed ad-hoc model. We assume that not all ad-hoc models show the same level of consistency as the presence of unexpected features and similar quality issues is a frequent problem (Leung & Bolloju, 2005). For a definite answer more comparisons between results of both methods are needed.

Completeness of our structured domain model did not surpass the ad-hoc model, due to two limitations. On the one hand the detail level of the models was not agreed upon before modeling as there was no actual business goal. Some of the missing details were present in the code system and could have been added. On the other hand, we worked with a given data set and could not fill gaps through further theoretical sampling. We suggest a case study that provides a practical task as well as access to data and stakeholders for interviews. The completeness of metamodel-based domain models should be re-evaluated and ways to extract fitting detail levels for different target groups need to identified.

In order to evaluate replicability people with different background and experience need to perform the procedure and the output needs to be evaluated. Our validation could not include this evaluation as only one person coded the data set. Other problems, for example, the increased effort and mistakes owed to manual transfers could be mitigated if the process was supported by appropriate software tools/CAQDAS features.

2.8 Conclusion

This thesis developed a metamodel which allows to derive structure and behavior models from one code system. Through iterative analysis steps the metamodel could be extended with two rule sets and suggestions for a fitting coding procedure. Validation showed that the resulting domain model was consistent and easily traceable to its original dataset. To leverage more advantages of the metamodel and the QDA process it is directed at, it will be necessary to conduct the previously suggested research, e.g. practical case studies. At least as important would be a sensible tool support for the application of the metamodel and for the transfer into different structural and behavioral models. We are convinced that further efforts like these will increase the applicability of the metamodel and contribute to its efficient use in requirements elicitation.

3 Elaboration Chapter

The research chapter on “A metamodel for code system” focused on providing the necessary information to understand our motivation, analysis and the resulting metamodel artifacts. Nevertheless, we would like to provide additional background information for various areas. The first sub-chapters can be used to get a more comprehensive understanding on which theoretical foundation we base our work. This comprises summaries on requirements engineering, domain models, QDA as well as metamodels and ontologies.

After that we will focus on practical considerations. By elaborating on our insights on the approach and the resulting procedure we explain which obstacles and problems still need to be overcome. One part of these problems can be mitigated by introducing new features or tool support for the method which we suggest in chapter 3.3. Others need to be addressed by further research as pointed out in chapter 3.4. Finally, we want to motivate both, new implementations and accompanying re-evaluations as well as further research by proposing expected benefits.

3.1 Theoretical Background

3.1.1 Requirements engineering

3.1.1.1 Requirements

The term “requirements” is used for a variety of information. Wiegers and Beatty (2013) compiled a list including user and system requirements, business rules, constraints, features, quality attributes and the common differentiation between functional and non-functional requirements. The variance of requirements originates in the variance of stakeholders most systems have (Wiegers & Beatty, 2013).

Across literature the differentiation between functional and non-functional requirements is most common and it is a challenge to integrate both perspectives when observing and modeling complex information systems (Doerr, 2013). Functional requirements describe the behavior of the system - what the system should do – in various situations. The definition becomes less distinct for non-functional requirements. Cost, performance, maintainability and robustness are examples named by Chung, Nixon, Yu and Mylopoulos (2012). More general non-functional requirements are characteristics and properties (Wiegers & Beatty, 2013) and can contain qualitative and quantitative criteria. Although they can be subjective and relative to the system considered, non-functional requirements are used as selection criteria for, the often vast amount, of software/functionality available (Chung, Nixon, Yu and Mylopoulos, 2012). Doerr (2013) supports this view. Before the right software solution can be selected, requirements need to be defined on the right detail level.

3.1.1.2 Activities of requirements engineering

The first step of developing requirements or requirements engineering is requirements elicitation. To discover or elicitate requirements is not easy (Laplante, 2013) and often methods from the social sciences like interviews and methods are facilitated (Macaulay, 2012). One example for the complexity of the elicitation process is the research by Browne and Rogish (2001) where a specific method to foster the information flow from the user to the analyst is developed. Requirements elicitation is about understanding the problem, whereas the later activities are about providing a solution to the identified problem (Robertson & Robertson, 2012).

However, before the requirements can be formed to a solution and modeled, analysis and reconciliation is necessary (Laplante, 2013). Statements from users and other stakeholders are not always logical and need to be adjusted. Formal methods applied during requirements elicitation can help to attain a more logical structure from the beginning. Additional improvements are achieved through an iterative analysis (Macaulay, 2012).

When it comes to the activity of modeling Rupp (2014) claimes that drafts are helpful for the development of requirements. More formal, Cardoso, Almeida & Guizzardi (2009) state that process models which allow a consistent and detailed understanding of a domain have a positive effect on the specification of business requirements. After the modeled requirements are validated and verified the process is not finished at this step. Requirements engineering could be seen as a process by which one only arrives at requirement documents (Macaulay, 2012) but the change over time should be taken into account by an ongoing requirements management (Laplante 2013).

A motivation to pay attention to all these requirements engineering activities is that it can reduce work in later phases, improve software qualities (Laplante, 2013) and independent from the development process it was found to be of high importance to the project success (Rupp, 2014). Laplante (2013) cites various studies that show that more effort should be put in requirements engineering which aligns with Fernandez and Wagner (2013) and the Standish Group (2004) who found that incomplete, inconsistent and overall poor requirements still are a major problem for software projects.

3.1.2 Domain model

The previous chapter makes clear that requirements engineering involves the assembly of information about the application domain to specify important non-functional requirements. Software systems are tied to their environment and become more closely related to it (Broy, 2013; Rausch et al., 2013). Therefore, domain models and the comprehensive knowledge they provide are highly relevant for software and system development (Broy, 2013).

A domain is the business area undergoing analysis (Daoust, 2012) and a domain model is the representation of all the relevant parts of this domain (Broy, 2013). Items important to the business are typically the people, organizations, places, things and events (Daoust, 2012). To represent these a domain model includes concepts, data types, functions, rules and laws as well as terminology and ontology rules (Broy, 2013). It identifies fundamental business and application specific entity types and relationships between them, including business processes. This overlaps with models of software systems in general. Models consist of notational and descriptive elements, and their structural and behavioral relationships (Rausch et al., 2013). Consequently, domain models like other models of software systems need to deal with problems related to complexity and scale. The trade-off as it is described by Rausch et al. (2013) is between accuracy of the results and the complexity of the model. While complex models are hard to analyze, understand and process, less complex models may be less relevant to the real system (Rausch et al., 2013). Despite the complexity domain models may reach, it needs to be said that they are still conceptual and solution independent (Larman, 2012).

Domain models are created during the early phases of the system development, which means they are an artefact of the requirements engineering like described previously. Model building, in conformity to the requirements engineering, is the process of capturing the experience and knowledge of an organization into reusable models (Heidrich, 2013). A class diagram, as one possibility to depict a domain, shows classes and interdependencies as representation of the real world structures. The challenge lies in extracting and analyzing the data from the requirements elicitation in a way to arrive at model conforming to the real world system.

3.1.3 Qualitative data analysis

We already highlighted the importance of qualitative data input for requirements and their corresponding domain. Therefore, we now want to outline by which means qualitative data can be analyzed.

QDA is an umbrella term for various methods that focus on the investigation and analysis of qualitative data. A wide range of interpretations of the term lead Kuckartz (2014) as well as Denzin and Lincoln (2011) to state that no common definition for QDA respectively qualitative research can be found. Even the definition of qualitative data is quite diverse. It can include test, images, audio-recordings, cultural artefacts and many more data sources (Kuckartz, 2014). Citing Sarker (2007), Chakraborty et al. (2015) point out that qualitative research methods can be classified as data-centric and data-driven which makes them a good choice for the data-centric context of requirements engineering and the data-driven system specifications in software engineering. Originally, however, QDA is a discipline from the social sciences

It is there that researchers often focus on the data collection and research design, whereas tangible and concrete method descriptions are hard to find (Kuckarz, 2014). Various mixed methods, which combine qualitative with quantitative research approaches, exist (e.g. Mayring interpretation method (Mayring, 2014)). Our work, however, focuses on requirements elicitation which is often done via interviews (Brown & Rogich, 2001) and on qualitative evaluations of named data. In grounded theory the considered data is qualitative and also the analysis method is qualitative (Kuckartz, 2014) making it a fitting method for our purpose.

3.1.3.1 Grounded theory method

Charmaz (1996) defined grounded theory method as "a logically consistent set of data collection and analytical procedures aimed to develop a theory". Although it was developed by Glaser and Strauss (1967) to examine interactions of humans in a social setting it can also be applied to gain insights about the interactions between humans and technical elements. This is supported by descriptions of grounded theory as a broad approach and a means to identify patterns for conceptual descriptions (Starks & Trinidad, 2007).

The central concept of the grounded theory method is the careful coding of the data (Kuckartz, 2014; Starks & Brown Trinidad, 2007)). Coding means to assign codes to specific phenomena in the data material. Three types of coding are differentiated: open, axial and selective coding.

Open Coding is a detailed examination of the data, and in parallel creating codes for all relevant phenomena. To ensure a critical view of the data Charmaz (1996) suggests the analyst to ask the following questions:

- What process is at issue here?
- Under which conditions does this process develop?
- How does the research participant think feel and act while involved in this process?
- When why and how does this process change?
- What are the consequences of the process?

These questions show that this first conceptualization has a strong focus on behavioral/process elements which can be used to define domains and software systems.

Axial Coding has a more structural approach. The data is reassembled into groupings based on relationships and patterns within and among the categories identified in the data.

Selective Coding is described in two ways. One is focusing on theory building, which means identifying and describing the central phenomenon for theory building (e.g. Starks & Brown Trinidad, 2007). The other view again focuses on the structure. Chakraborty & Dehlinger (2009) suggest to construct relationships, including sequence descriptions between the higher order categories that were identified during axial coding.

In parallel to the coding the analyst can use memos to keep track of issues to be addressed in the next iteration and to start theory building. After the initial data collection and analysis new data is collected to fill the gaps and answer open questions. This concept is named theoretical sampling and is continued until the theory is completed – theoretical saturation is reached.

Although the defined goal of grounded theory analysis is to produce a theory, some analysts identify patterns only within and between categories which results in conceptual descriptions (Stark & Brown Trinidad, 2007). This is favorable for building domain models, but one has to be careful to declare that only steps of the grounded theory method are used and no complete grounded theory approach was conducted (Denzin & Lincoln, 2011).

3.1.4 Metamodels and ontologies

For our purpose of creating models that support the requirements engineering process we designed a metamodel that would structure the output of QDA for the use in models. As Flower and Edeki (2013) stated formal models are crucial for the modeling process. A metamodel provides this formality because it unifies the language and the construction process (Clark, Sammut & Williams, 2008).

Metamodels and ontologies are, in the broadest sense, used to provide guidelines for the construction of (language) systems. In both cases specific elements and their relations are defined. They are represented in the form of UML classes, which again are arranged in a subclass – superclass hierarchy as suggested by Noy and McGuiness (2001) and Guizzardi (2007). For further orientation we used the following definitions:

A metamodel is a description of the languages abstract syntax since it defines

- a set of constructs selected for the purpose of performing a specific set of tasks and
- a set of well-formedness rules for combining these constructs in order to create grammatically valid models in the language (Guizzardi, 2007).

An ontology is a theory concerning the kinds of entities and specifically the kinds of abstract entities that are to be admitted to a language system (Webster Dictionary)

Nissen, Jeusfeld, Jarke, Zemanek and Huber (1996) suggest to use a metamodeling tool to transfer qualitative data input into various parts of requirement documents. They describe that “while requirement models are abstract representations of an existing or desired real world, metamodels are abstract representations of existing or desired requirement models and their interrelationships” (Nissen et al., 1996). During our analysis and evolution of our metamodel we constantly took into account two statements by Schuette and Rotthowe (1998). “A model is complete according to the metamodel if the relationships between the information objects described in the metamodel are applied in the model itself in the same way”; “A model is consistent with the metamodel, if the included information objects in the model are completely defined in the meta model.” Once the abstract syntax is defined additional rules need to be generated to ensure the right application of the metamodel (e.g. Clark et al., 2008).

3.2 Insights on Approach and Procedure

In general, we realized that the coding and modeling should not be done by only one person due to several resulting bias. On the one hand every person has a specific focus influencing the analysis. The different perspective and "Weltanschauung" of analysts causes them to choose different ways of conceptual development (Schuette & Rotthowe, 1998). This became obvious during the expert interviews which revealed that the ad-hoc model contained more of the technical elements concerning Inner Source whereas the metamodel-based model contained more social elements. Had both analysts conducted the coding with their focus, a more complete representation of the domain would have been possible. The other bias, could also be mitigated through intercoder agreements and multiple analysts. It is the choice of names and prioritizing for Categories and Concepts as well as Properties and Relationships.

Another general drawback was the manual transfer of codes into the domain model. One generalization of Inner Source was missed (12# Infrastructure-based Inner Source) and previously correctly named codes became incorrect through typing errors.

3.2.1 Scope and detail level

Our decision to stick with a low detail level (e.g. summarizing all aspects which describe possible benefits of Inner Source and motivations of companies to use Inner Source under the term "Motivation" instead of naming the single concepts; leaving out detailed process steps) had a negative impact on two modeling steps. The chosen names given to the codes were not always self-explanatory and as sub-codes were missing and no glossary was written up due to time constraints the understanding of some domain model elements was hindered. Furthermore, not many behavioral aspects could be structured in terms of sequence. This, however, was additionally influenced by the domain and the given data set.

A general problem of our method we noted during the Inner Source coding is that it is hard to distinguish between the as-is state and possible to-be states. There are two main approaches when modeling development processes: Prescriptive modeling defines how development activities should be done, whereas descriptive modeling defines how development activities are actually performed in an organization (Becker-Kornstaedt & Belau, 2000). Depending on the purpose of the domain model one should actively decide for one or both states at the beginning of the modeling process.

3.2.2 Word choice in coding

At the beginning codenames were chosen more strictly to fit classes, attributes or activities but it became apparent that we could not describe the domain exclusively with such single terms. For example, "faster development" is a benefit highlighted throughout the sources but as we thought in attributes the label became "development speed". Due to that "faster development" cannot be identified as an essential part of Inner Source when looking at the domain model. Other constructs proofed difficult as well. Schmitt (2015, 2016) has used phrases containing more than one concept making it easier to identify sequences and decisions but harder to single out classes. To explicitly code relations as we did in the "need access"-example (relation between Stakeholder #16 and Code #44) was not helpful as labeling the participating parties in the code system would have conflicted with our metamodel.

Although we suggest that codes for structural aspects should contain a noun and codes for dynamic aspects should contain a verb we think that an appropriate word choice to balance capturing the concepts and representing them as element of the metamodel will need further evaluation.

One strict rule we would implement, if a rule set for word choice is developed, is to choose the singular form of nouns. As they later represent classes and therefore blueprints for several instances the plural form would be misleading.

3.2.3 Handling of different data sources

The data set on Inner Source contained many research papers which forced two decisions. Abstracts, conclusions and suggestions for further research were not coded except new (in-vivo) codes were introduced. Repetitive explanations without new insights need to be coded as well to distinguish between more and less relevant codes. The second decision can be scrutinized because it could be that aspects of high relevance may only be explained by one author while minor aspects are constantly stated without having a big impact on the domain.

Especially problematic is the handling of literature reviews which involve papers also included in the data set. Coding aspects in these chapters could also create the wrong impression of relevance for basic constructs.

If, as in our case, the starting point is not a single source but a data set the order of processing could have an influence on word choice and some structural aspects. The first iteration of open and axial coding may bias the overall outcome and hinders replicability. We therefore suggest to mix authors, type of source and publication year. Additionally, the order of sources should be changed if a second analysts works on the data set to further reduce bias.

3.2.4 Mapping rules and modeling

In general, the manual modeling is an unnecessary effort as the mapping rules would allow an automatic transfer. However, we identified one missing concept which we unfortunately could not test sufficient to include it in the metamodel and rule system. For behavioral models one would not only need to depict parallel processes/activities but also decisions. By introducing a condition as attribute in the *causes* element (e.g. “create patch” causes “commit patch (developer has commit bid)”; “ask for integration (developer has no commit bid)”) one could model decisions. Another possibility would be to introduce “Decision” as specialization of Activity and integrate it in the existing is consequence/causes sequence.

3.3 Suggestions for Features and Tool Support

As there are various possibilities to create tool support for our concept the suggestions are just roughly outlined. We believe that some simple features which could be implemented in existing CAQDAS like MaxQDA could overcome some of the obstacles named in the previous chapters.

First and foremost, every code needs additional data fields that can be chosen by the user. To ideally support the process, we suggest these data fields should be set to the elements of the metamodel as a default. Alterations and additions through the user should be possible. As soon as these data fields exist automated entries can be created to reduce the work load for the analyst.

Example 1: Employee is related to Performance Management Cycle

- As soon as the "is related to" data field of "Employee" is filled with "Performance Management Cycle" the corresponding entry ("Employee") is created in the "is related to" data field of "Performance Management Cycle".
- Additional to creating the entry a pop-up window can ask if the user wants to add a name to the association.
- If the code which is entered in the "is related to" data field does not exist a warning is displayed and the user can choose to discard the entry or create the code.

Example 2: text contains the description of a relationship like "feedback is tool based"

- Condition: The objects/classes "Feedback" and "Software tools" already exist.
- Instead of adding a code "feedback is tool based" the user can choose the function "Add relationship"
- A pop-up window then offers to choose
    - starting and end point from a dropdown of existing classes (e.g. "Feedback", "Software tools")
    - drop down with available relationship types (e.g. "is related to")
    - an input field for the name of the relationship is activated if it's an association.
- After saving this input the following entries would be created:
    - "is related to" data field of "Feedback": "Software tools (is based on)"
    - "is related to" data field of "Software tools": "Feedback"

To easily create use cases (via cross referencing Actor-codes and Process-/Activity-codes) and swimlanes we think that autocoding all appearances of identified roles would be extremely helpful. If codings could be manipulated via dragging it may also be reasonable to autocode other repetitive aspects.

If the analyst codes reoccuring sentences or phrases with different codes or if already existing codes are set up an indication should be displayed. This would help the sorting process during axial coding (no unintental duplicates that have to be merged).

To help the process of coding an OCR should be included because coding "Images" makes it harder to write up glossaries and gain a quick overview over codings. Clarity during coding-would also be improved if the codes in documents with two columns would be displayed on both sides of the document.

3.4 Further Research

One criteria we could not evaluate was the replicability of the domain model. Therefore, we suggest that two independent analysts set out to model the same domain, with the same access to data sources or respectively interview partners, and the resulting models are compared for differences. The same should be done with a setting where two analysts cooperate on the coding process.

Exploratory research is necessary to include decisions in the metamodel and to identify fitting word choices for the code systems. Additionally, various cooperation models could be tested.

To further validate the metamodel it should be combined with other modeling languages. It may be possible to create a more process oriented overview of the domain by transferring the elements via BPMN.

All further research should re-evaluate the procedure as soon as the tool support is enhanced.

3.5 Expected Benefits

Besides the advancements in the domain models created (consistency, completeness, transparency and replicability) we expect further benefits along the road. They depend on the outcome of future research as well as on the implementation of tool support.

The structured approach forces requirements engineers to run an in-depth analysis of the domain, its processes and the target system. In traditional requirements engineering, time pressure or other constraints would typically lead to a superficial analysis, neglecting some important complex aspects. The deficiencies may only be discovered in late phases where they already have caused errors. The analysts which went through the coding steps can serve as experts during the project and may be suitable as product owner in Scrum, member of the core team for Inner Source development or project manager in traditional development.

Our approach allows sensible access to the source data which can be used as a wiki to support on-boarding of new team members or retracing discussions and arguments from the beginning in late phases of the project. The negative aspect of the analyst/expert leaving the team can also be mitigated as the information basis is easy accessible and decisions can be retraced.

During requirements management the through the code system well-structured data-basis, allows to identify possible contradictions of new requirements or change requests with previous statements. If rejections or decisions are challenged the drill-down in the data allows to identify involved parties and their arguments and therefore speeds up resolving the conflict.

Appendices

Appendix A: Interview Structure

- Please state your opinion about the following statements in regard to the domain model presented to you.
- First, please state how much you agree with the statement on a scale from one to seven – one representing that you fully agree and seven that you fully disagree.
- After this please give reason for your rating.

1. The conceptual schema represents the domain correctly.

2. All the elements in the conceptual schema are relevant for the representation of the domain.

3. The conceptual schema gives a complete representation of the domain.

4. Elements must be added to faithfully represent the domain.

5. The conceptual schema contains redundant elements.

6. The conceptual schema contains contradicting elements.

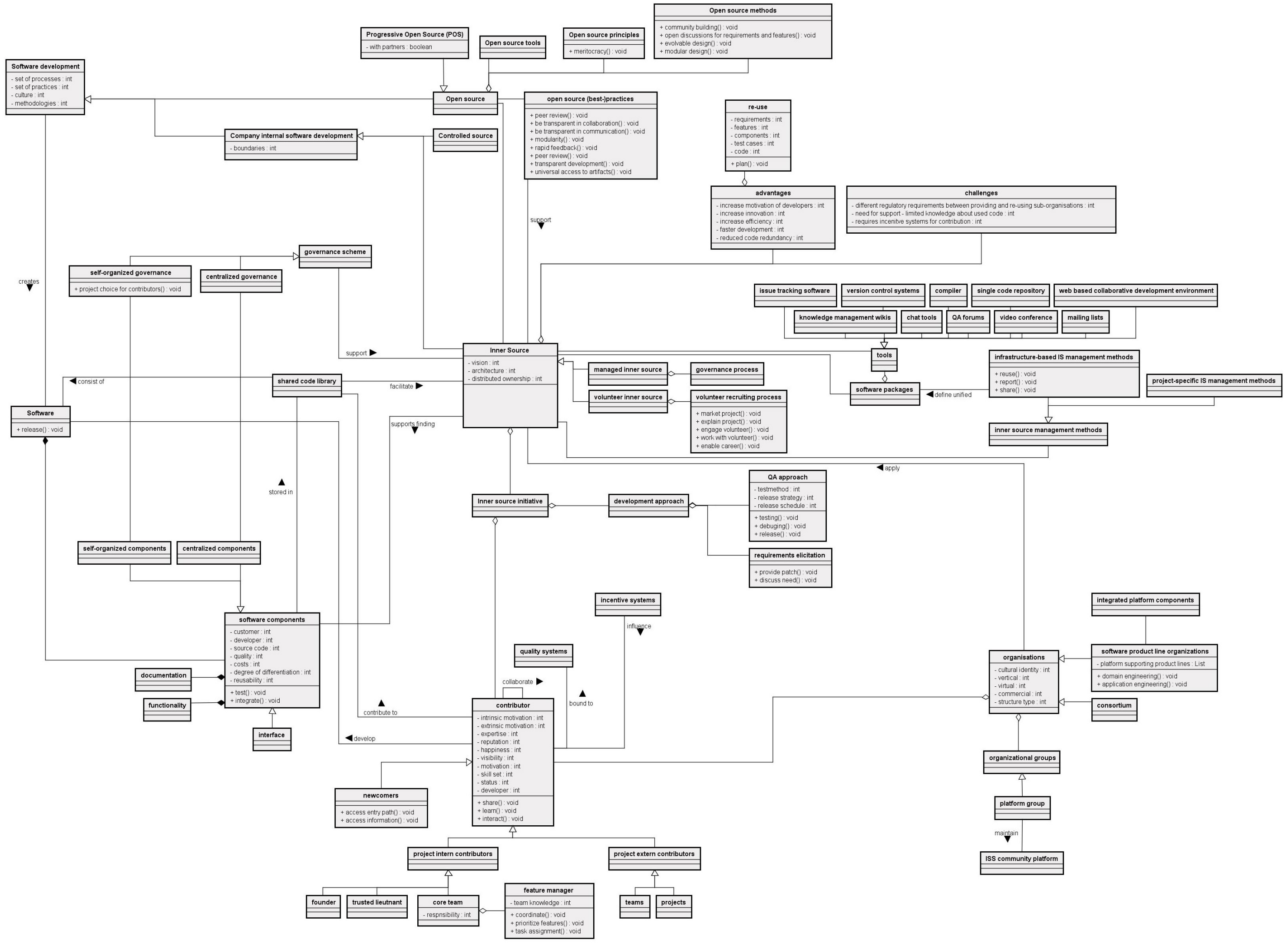
7. The conceptual schema is a realistic representation of the domain.

8. Overall, I think the conceptual schema would be an improvement to a textual description of the domain.

9. Overall, I found the conceptual schema useful for understanding the domain.

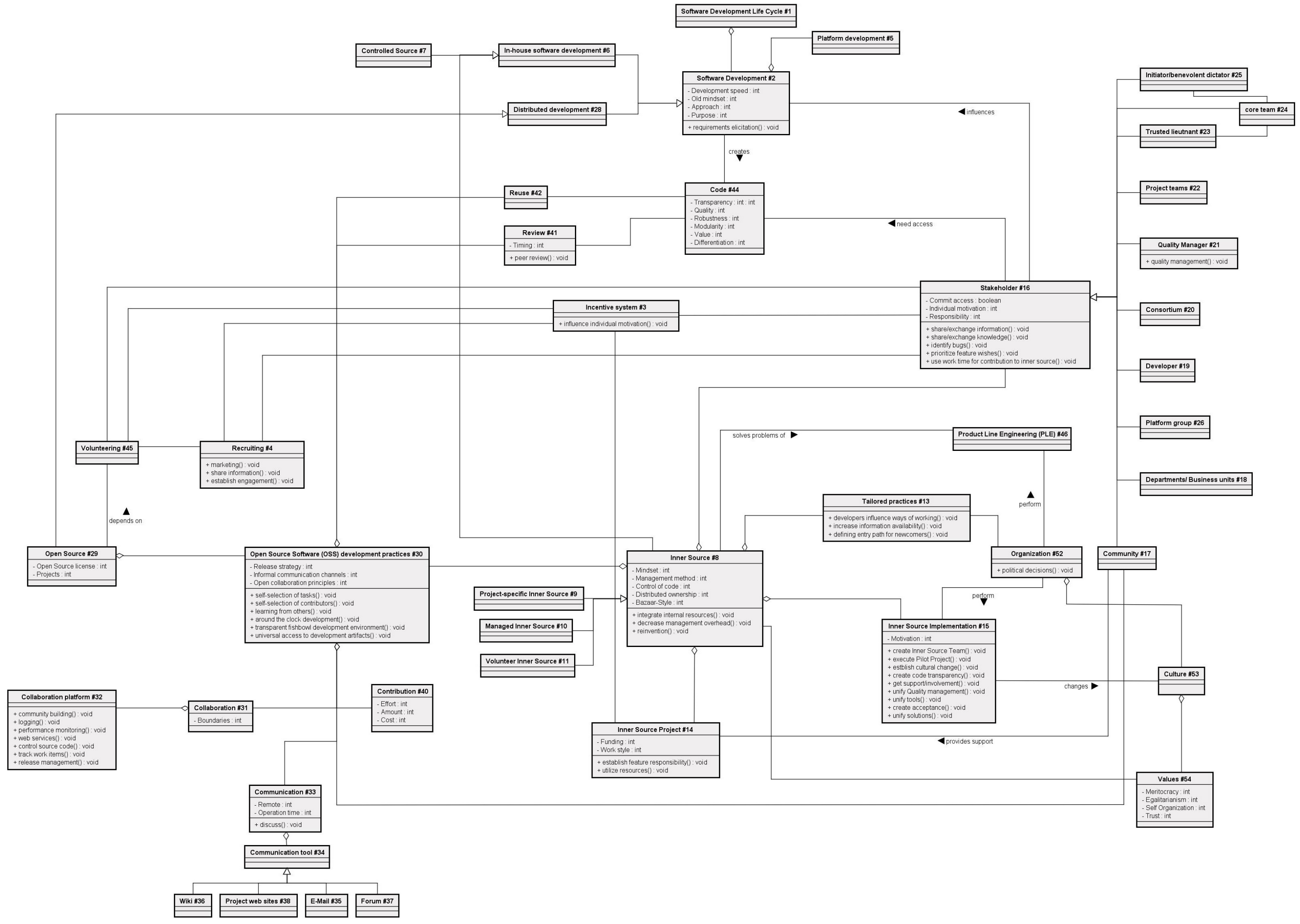
10. Overall, I think the conceptual schema would improve someone's performance when understanding the domain/my performance in explaining the domain to someone not familiar with it.

Appendix B: Domain Model Inner Source (Ad-hoc)



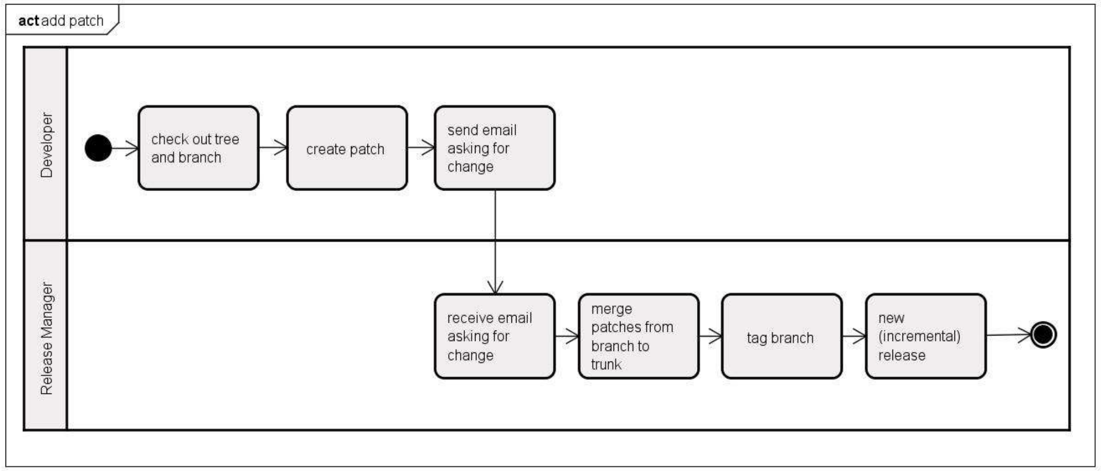
Ad-hoc domain model Inner Source

Appendix C: Domain Model Inner Source (Metamodel-based)


Software Development Life Cycle #1
Platform development #5
Controlled Source #7
In-house software development #6
Distributed development #28
Software Development #2
- Development speed : int
- Old mindset : int
- Approach : int
- Purpose : int
+ requirements elicitation() : void
influences
creates
Reuse #42
Review #41
- Timing : int
+ peer review() : void
Code #44
- Transparency : int : int
- Quality : int
- Robustness : int
- Modularity : int
- Value : int
- Differentiation : int
need access
Initiator/benevolent dictator #25
core team #24
Trusted lieutnant #23
Project teams #22
Quality Manager #21
+ quality management() : void
Stakeholder #16
- Commit access : boolean
- Individual motivation : int
- Responsibility : int
+ share/exchange information() : void
+ share/exchange knowledge() : void
+ identify bugs() : void
+ prioritize feature wishes() : void
+ use work time for contribution to inner source() : void
Incentive system #3
+ influence individual motivation() : void
Consortium #20
Developer #19
Platform group #26
Departments/ Business units #18
Volunteering #45
Recruiting #4
+ marketing() : void
+ share information() : void
+ establish engagement() : void
solves problems of
Product Line Engineering (PLE) #46
perform
Tailored practices #13
+ developers influence ways of working() : void
+ increase information availability() : void
+ defining entry path for newcomers() : void
depends on
Open Source #29
- Open Source license : int
- Projects : int
Open Source Software (OSS) development practices #30
- Release strategy : int
- Informal communication channels : int
- Open collaboration principles : int
+ self-selection of tasks() : void
+ self-selection of contributors() : void
+ learning from others() : void
+ around the clock development() : void
+ transparent fishbowl development environment() : void
+ universal access to development artifacts() : void
Inner Source #8
- Mindset : int
- Management method : int
- Control of code : int
- Distributed ownership : int
- Bazaar-Style : int
+ integrate internal resources() : void
+ decrease management overhead() : void
+ reinvention() : void
Organization #52
+ political decisions() : void
Community #17
Project-specific Inner Source #9
Managed Inner Source #10
Volunteer Inner Source #11
perform
Inner Source Implementation #15
- Motivation : int
+ create Inner Source Team() : void
+ execute Pilot Project() : void
+ estblish cultural change() : void
+ create code transparency() : void
+ get support/involvement() : void
+ unify Quality management() : void
+ unify tools() : void
+ create acceptance() : void
+ unify solutions() : void
changes
Culture #53
Collaboration platform #32
+ community building() : void
+ logging() : void
+ performance monitoring() : void
+ web services() : void
+ control source code() : void
+ track work items() : void
+ release management() : void
Collaboration #31
- Boundaries : int
Contribution #40
- Effort : int
- Amount : int
- Cost : int
Inner Source Project #14
- Funding : int
- Work style : int
+ establish feature responsibility() : void
+ utilize resources() : void
provides support
Values #54
- Meritocracy : int
- Egalitarianism : int
- Self Organization : int
- Trust : int
Communication #33
- Remote : int
- Operation time : int
+ discuss() : void
Communication tool #34
Wiki #36
Project web sites #38
E-Mail #35
Forum #37

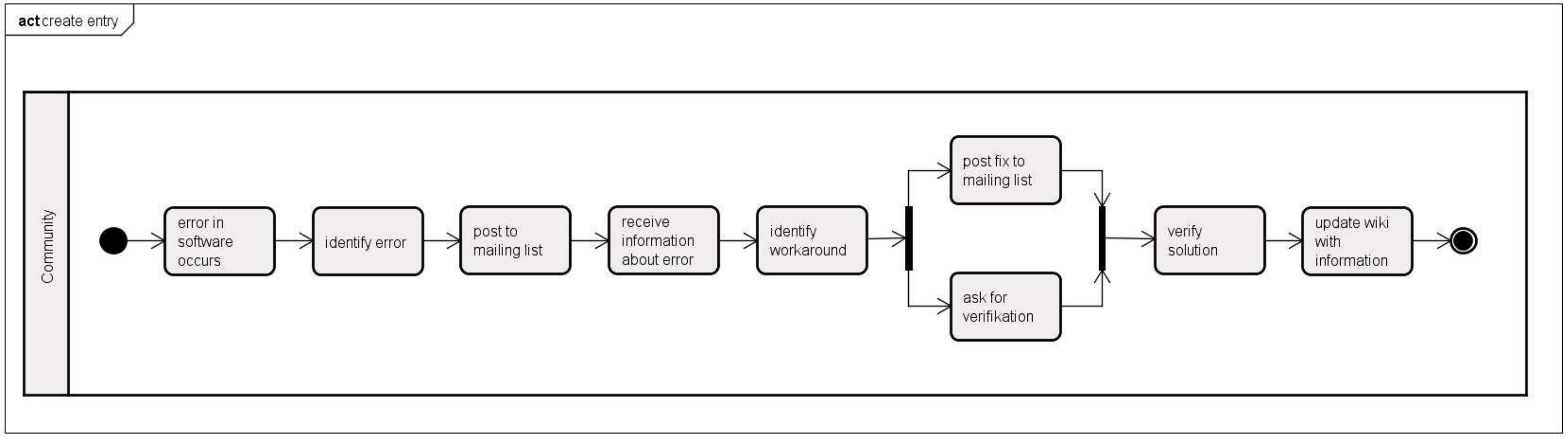
Domain model Inner Source (metamodel-based / structural part)

Appendix D: Behavioral Models Inner Source



act add patch
Developer
Release Manager
check out tree and branch
create patch
send email asking for change
receive email asking for change
merge patches from branch to trunk
tag branch
new (incremental) release

exemplary process "add patch"



exemplary process "create entry"

Appendix E: Mapping Rules

Tables E1 and E2 show exemplary codes and their assigned metadata. The column content is color coded in order to ease readability/transfer of the mapping rules presented in tables E3 and E4. The rules presented refer to notations of an UML class diagram (Table E3) and an UML Activity diagram (Table E4). In the pseudocode the following constructs are used:

- "FOR EACH xyz" means that the rules need to be applied to all xyz before the next step can start.
- Relationship cells can contain more than one value. The values are separated by semicolons. "FOR EACH VALUE" refers to the single values in the named cell.
- If something is referred to in quotation marks the content of this field should be inserted (e.g. classname = "code" -> classname = Distributed Development).
- "GET xyz WHERE code = = abc" to refer to a selection from the code system
- "origin WHERE"/"origin" to select and set the starting point of a relationship/its graphical representation
- "target WHERE"/"target" to select and set the end point of a relationship/its graphical representation, respectively the process frame a sub-process/activity belongs to.

code	**Label**	**Aspect**	**Relationship**		
			is a	*is part of*	*is related to*
Distributed Development	*Category*	*Object*			
Open Source	*Category*	*Object*	Distributed Development		Volunteering (depends on)
Volunteering	*Category*	*Object*			Recruiting; Incentive system
OSS development practices	*Concept*	*Object*		Open Source	
OS license	*Property*	*Object*			Open Source
GPL	*Property*	*Object*	OS license		

Table E1: Examples for codes with metamodel information (part 1)

code	**Label**	**Aspect**	**Relationship**			
			causes	is cons. of	performs	influences
Community	*Concept*	*Actor*			create wiki; […]; Inner Source Project (provides support)	
create entry	*Property*	*Process*		create wiki		Wiki
error in software occurs	*Property*	*Activity*	identify bug	START		create wiki
identify bug	*Property*	*Activity*	post to mailing list	error in software occurs		create wiki
post to mailing list	*Property*	*Activity*	identify workaround	identify bug		create wiki
identify workaround	*Property*	*Activity*	post fix to mailing list; ask for verifikation	post to mailing list		create wiki
post fix to mailing list	*Property*	*Activity*	verify solution	identify workaround		create wiki
ask for verification	*Property*	*Activity*	verify solution	identify workaround		create wiki
verify solution	*Property*	*Activity*	update wiki with information about fix	ask for verifikation; post fix to mailing list		create wiki
update wiki with information about fix	*Property*	*Activity*	END	verify solution		create wiki

Table E2: Examples for codes with metamodel information (part 2)

Metamodel information of the code	Structural domain model
FOR EACH code {	

IF **Label** = = (*Category* OR *Concept*)	CREATE class
	{classname = "code"}

}
FOR EACH code {	

IF **Label** = = (*Category* OR *Concept*)
AND *is a* IS NOT EMPTY
	CREATE generalization
	{origin WHERE classname = = "code";
	target WHERE classname = = "is a"}

IF **Label** = = (*Category* OR *Concept*)
AND *is part of* IS NOT EMPTY
	CREATE aggregation
	{origin WHERE classname = = "code";
	target WHERE classname = = "is part of"}

IF **Label** = = (*Category* OR *Concept*)
AND is related to IS NOT EMPTY
	FOR EACH VALUE {CREATE association
	{origin WHERE classname = = "code";
	target WHERE classname = = "is related to";
	IF "(name)" EXISTS IN *is related to*
	{associationname = "name"}}}

IF **Label** = = *Property*
AND **Aspect** = = (*Object* OR *Actor* OR *Place*)
AND is related to IS NOT EMPTY
	CREATE attribute
	{target WHERE classname = = "is related to";
	propertyname = "code"}

IF **Label** = = *Property*
AND *is a* IS NOT EMPTY
	CREATE value FOR attribute
	{target WHERE classname = = "is a";
	valuename = "code"}

IF **Label** = = *Property*
AND **Aspect** = = (*Activity* OR *Process*)
AND influences IS NOT EMPTY
	CREATE method
	{target WHERE classname = = "influences";
	methodname = "code"}

IF **Label** = = (*Category* OR *Concept*)
AND **Aspect** = = *Actor*
AND *performs* IS NOT EMPTY
	FOR EACH VALUE {
	GET **Label** WHERE code = = "performs"
	IF **Label** = = *Property*
	{DO NOTHING}
	ELSE {CREATE association
	{origin WHERE classname = = "code";
	target WHERE classname = = "performs";
	IF "(name)" EXISTS IN *performs*
	{associationname = "name"}
	ELSE {associationname = performs}}}}

}	

Table E3: Mapping rules - structural model

Metamodel information of the code	Behavioral domain model
FOR EACH code {	

IF **Aspect** = = *Activity*	CREATE activity
	{activityname = “code”}

IF **Aspect** = = *Process*	CREATE process
	{processname = “code”}

}
FOR EACH process {
GET “influences” AS targetprocess WHERE code = = processname;
GET **Aspect** AS targetaspect WHERE code = = targetprocess;
GET “code” AS actorname WHERE processname IN “performs”;	

IF ((*influences* IS EMPTY)
OR targetaspect = = (*Object* OR *Actor* OR *Place*))
AND actorname EXISTS
	CREATE swimlane
	{target = = process;
	swimlanename = actorname}

IF targetaspect = = *Process*
AND actorname EXISTS
	IF swimlanename = = actorname EXISTS IN tar-getprocess
	{MOVE process INTO swimlane}
	ELSE {{CREATE swimlane
	{target WHERE processname = = targetprocess;
	swimlanename = actorname}}
	{MOVE process INTO swimlane}}

IF targetaspect = = *Process* AND actorname NOT EXISTS	{MOVE process INTO targetprocess}

}
FOR EACH activity {
GET “influences” AS targetprocess WHERE code = = activityname;
GET **Aspect** AS targetaspect WHERE code = = targetprocess;
GET “code” AS actorname WHERE activityname IN “performs”;	

IF targetaspect = = *Process*
AND actorname EXISTS
	IF swimlanename = = actorname EXISTS IN tar-getprocess
	{MOVE activity INTO swimlane}
	ELSE {{CREATE swimlane
	{target WHERE processname = = targetprocess;
	Swimlanename = actorname}}
	{MOVE activity INTO swimlane}}

IF targetaspect = = *Process* AND actorname NOT EXISTS	{MOVE activity INTO targetprocess}
…	

Metamodel information of the code	Behavioral domain model

}
FOR EACH process {
COUNT VALUES of is consequence of AS incoming WHERE code = = processname;
COUNT VALUES of *causes* AS outgoing WHERE code = = processname;	

IF is consequence of = = START	CREATE startnode;
	CREATE transition
	{origin = = startnode; target = = process}

IF incoming = = 1	CREATE transition
	{origin = = WHERE processname = = is conse-quence of; target = = process}

IF incoming > 1	CREATE merge
	{FOR EACH {CREATE transition
	{origin = = “is consequence of”;
	target = = merge}};
	CREATE transition
	{origin = = merge; target = = process}

IF *causes* = = END	CREATE endnode;
	CREATE transition
	{origin = = process; target = = endnode}

IF outgoing = = 1	CREATE transition
	{origin = = process;
	target WHERE processname = = “causes”}

IF outgoing > 1	CREATE fork
	{FOR EACH {CREATE transition
	{origin = = fork;
	target WHERE processname = = “causes”}};
	CREATE transition
	{origin = = process; target = = fork}

}
FOR EACH activity {
// see process: exchange processname with activityname
}
ERASE DUPLICATES;	

Table E4: Mapping rules - behavioral model

References

Agarwal, R., & Tanniru, M. R. (1990). Knowledge acquisition using structured interviewing: an empirical investigation. *Journal of Management Information Systems*, *7*(1), 123-140.

Becker, J. (2 011). Information Models for Process Management–New Approaches to Old Challenges. In *Emerging Themes in Information Systems and Organization Studies* (pp. 145-154). Physica-Verlag HD.

Becker-Kornstaedt, U., & Belau, W. (2000). Descriptive process modeling in an industrial environment: Experience and guidelines. In *European Workshop on Software Process Technology* (pp. 176-189). Springer Berlin Heidelberg.

Binder, M., & Edwards, J. S. (2010). Using grounded theory method for theory building in operations management research: A study on inter-firm relationship governance. *International Journal of Operations & Production Management*, *30*(3), 232-259.

Browne, G. J., & Rogich, M. B. (2001). An empirical investigation of user requirements elicitation: Comparing the effectiveness of prompting techniques. *Journal of Management Information Systems*, *17*(4), 223-249.

Broy, M. (2013). Domain Modeling and Domain Engineering: Key Tasks in Requirements Engineering. In *Perspectives on the Future of Software Engineering* (pp. 15-30). Springer Berlin Heidelberg.

Cardoso, E. C., Almeida, J. P. A., & Guizzardi, G. (2009). Requirements engineering based on business process models: A case study. In *EDOCW* (pp. 320-327).

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.

Chakraborty, S., & Dehlinger, J. (2009). Applying the grounded theory method to derive enterprise system requirements. In *Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, 10th ACIS International Conference on* (pp. 333-338). IEEE.

Chakraborty, S., Rosenkranz, C., & Dehlinger, J. (2015). Getting to the shalls: Facilitating sensemaking in requirements engineering. *ACM Transactions on Management Information Systems (TMIS)*, *5*(3), 14.

Charmaz, K. (1996). The search for Meanings – Grounded Theory. In *Rethinking Methods in Psychology*. (pp. 27-48). London: Sage Publications.

Cherfi, S. S. S., Akoka, J., & Comyn-Wattiau, I. (2002). Conceptual modeling quality-from EER to UML schemas evaluation. In *International Conference on Conceptual Modeling* (pp. 414-428). Springer Berlin Heidelberg.

Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2012). *Non-functional requirements in software engineering* (Vol. 5). Springer Science & Business Media.

Clark, T., Sammut, P., & Willans, J. (2008). Applied metamodelling: a foundation for language driven development.

Coleman, G., & O'Connor, R. (2007). Using grounded theory to understand software process improvement: A study of Irish software product companies. *Information and Software Technology*, *49*(6), 654-667.

Corbin, J. M., & Strauss, A. (1990). Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology*, 13(1), 3-21.

Daoust, N. (2012). *UML Requirements Modeling for Business Analysts: Steps to Modeling Success*. Technics Publications.

Davison, R., Martinsons, M. G., & Kock, N. (2004). Principles of canonical action research. *Information systems journal*, *14*(1), 65-86.

Denzin, N. K., & Lincoln, Y. S. (2011). *The SAGE handbook of qualitative research*. Sage.

De Oca, I. M. M., Snoeck, M., Reijers, H. A., & Rodríguez-Morffi, A. (2014). A systematic literature review of studies on business process modeling quality. *Information and Software Technology*, *58*, 187-205.

Doerr, J. (2013). Modeling Complex Information Systems. In J. Münch & K. Schmid (eds.), *Perspectives on the Future of Software Engineering* (pp. 95-10). Berlin Heidelberg: Springer Verlag

Dorussen, H., Lenz, H., & Blavoukos, S. (2005). Assessing the reliability and validity of expert interviews. *European Union Politics*, *6*(3), 315-337.

Easterbrook, S., Singer, J., Storey, M. A., & Damian, D. (2008). Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering* (pp. 285-311). Springer London.

Fernandez, W. D., & Lehmann, H. (2011). Case studies and grounded theory method in information systems research: issues and use. *Journal of Information Technology Case and Application Research*, *13*(1), 4-15.

Fernandez, D. M., & Wagner, S. (2013). *Naming the Pain in Requirements Engineering–NaPiRE Report 2013*. Technical Report TUM-I1326, Technische Universität München.

Flowers, R., & Edeki, C. (2013). Business process modeling notation. *International Journal of Computer Science and Mobile Computing*, *2*(3), 35-40.

Geambasu, C. V. (2012). BPMN vs. UML Activity Diagram for business process modeling. *Accounting and Management Information Systems*, *11*(4), 637.

Glaser, B. G., & Strauss, A. L. (1999). *The discovery of grounded theory: Strategies for qualitative research*. New York: Aldine de Gruyter.

Green, P., & Rosemann, M. (2000). Integrated process modeling: an ontological evaluation. *Information systems*, *25*(2), 73-87.

Heidrich, J. (2013). Continuous Process Improvement. In J. Münch & K. Schmid (eds.), *Perspectives on the Future of Software Engineering* (pp. 111-129). Berlin Heidelberg: Springer Verlag

Indulska, M., Recker, J., Rosemann, M., & Green, P. (2009). Business process modeling: Current issues and future challenges. In *International Conference on Advanced Information Systems Engineering* (pp. 501-514). Springer Berlin Heidelberg.

Kaufmann, A., & Riehle, D. (2015). Improving Traceability of Requirements Through Qualitative Data Analysis. In *Software Engineering & Management* (pp. 165-170).

Kuckartz, U. (2014). *Qualitative text analysis: A guide to methods, practice and using software*. Sage.

Kunz, K. (2015), *Developing a Domain Analysis Procedure based on Grounded Theory Method.* Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg. Retrieved 15 May 2016 from http://osr.cs.fau.de/wpcontent/uploads/2015/06/kunz 2015 arbeit.pdf

Laplante, P. A. (2013). *Requirements engineering for software and systems*. CRC Press.

Larman, C. (2012). *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interactive Development*. Pearson Education India.

Leung, F., & Bolloju, N. (2005). Analyzing the quality of domain models developed by novice systems analysts. In *Proceedings of the 38th annual Hawaii international conference on system sciences* (pp. 188b-188b). IEEE.

List, B., & Korherr, B. (2006). An evaluation of conceptual business process modelling languages. In *Proceedings of the 2006 ACM symposium on Applied computing*. ACM. 1532-1539.

Locke, K. (2001). *Grounded theory in management research*. Sage.

Macaulay, L. A. (2012). *Requirements engineering*. Springer Science & Business Media.

Mayring, P. (2014). Qualitative content analysis: theoretical foundation, basic procedures and software solution.

McGraw, K. L. (1989). *Knowledge acquisition: principles and guidelines* (No. QA76. 76. E95 M61).

Milisterfer, O. (2016), *Comparing domain models developed with a grounded theory-based method and ad-hoc modeling.* Studienarbeit, Friedrich-Alexander-Universität Erlangen-Nürnberg.

Moody, D. L. (2005). Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions. *Data & Knowledge Engineering*, *55*(3), 243-276.

Nissen, H. W., Jeusfeld, M. A., Jarke, M., Zemanek, G. V., & Huber, H. (1996). Managing multiple requirements perspectives with metamodels. *IEEE Software*, *13*(2), 37.

OMG (2005). Documents associated with UML® Version 2.0. Retrieved from http://www.omg.org/spec/UML/2.0/

Panayiotou, N. A., Gayialis, S. P., Evangelopoulos, N. P., & Katimertzoglou, P. K. (2015). A business process modeling-enabled requirements engineering framework for ERP implementation. *Business Process Management Journal*, *21*(3), 628-664.

Poels, G., Maes, A., Gailly, F., & Paemeleire, R. (2005). Measuring the perceived semantic quality of information models. In *International Conference on Conceptual Modeling* (pp. 376-385). Springer Berlin Heidelberg.

Rausch, A., Bartelt, C., Herold, S., Klus, H., & Niebuhr, D. (2013). From software systems to complex software ecosystems: model-and constraint-based engineering of ecosystems. In *Perspectives on the Future of Software Engineering.* (pp. 61-80). Springer Berlin Heidelberg.

Robertson, S., & Robertson, J. (2012). *Mastering the requirements process: Getting requirements right*. Addison-Wesley.

Rupp, C. (2014). *Requirements-Engineering und-Management: Aus der Praxis von klassisch bis agil*. Carl Hanser Verlag GmbH Co KG.

Salow, S. (2016). *Using qualitative data analysis for business process modeling*. Studienarbeit, Friedrich-Alexander-Universität Erlangen-Nürnberg.

Sarker, S. (2007). *Qualitative research genres in the is literature: Emerging issues and potential implications*. In Proceedings of the 40th Annual Hawaii International Conference on System Sciences.

Schmitt, F. (2015). *Improving Domain Modeling And Requirements Analysis Using Grounded Theory.* Studienarbeit, Friedrich-Alexander-Universität Erlangen-Nürnberg. Retrieved 15 May 2016 from https://osr.cs.fau.de/wp-content/uploads/2015/06/schmitt_2015_arbeit.pdf

Schmitt, F. (2016). *Integrating Multiple Views In A Code System.* Diplomarbeit, Friedrich-Alexander-Universität Erlangen-Nürnberg. Retrieved 15 May 2016 from https://osr.cs.fau.de/wp-content/uploads/2016/02/schmitt_2016_arbeit.pdf

Schuette, R., & Rotthowe, T. (1998). The guidelines of modeling–an approach to enhance the quality in information models. In *International Conference on Conceptual Modeling* (pp. 240-254). Springer Berlin Heidelberg.

Seidl, M., Scholz, M., Huemer, C., & Kappel, G. (2015). *UML@ classroom: An introduction to object-oriented modeling*. Springer.

Standish, The Standish Group International, Inc. (2004), *Third Quarter Research Report*, West Yarmouth, MA.

Starks, H., & Brown Trinidad, S. B. (2007). Choose your method: A comparison of phenomenology, discourse analysis, and grounded theory. *Qualitative health research*, 17(10), 1372-1380.

Webster Dictionary. Retrieved 16 September 2016 from http://www.webster-dictionary.org/definition/Ontology

Wiegers, K., & Beatty, J. (2013). *Software requirements*. Pearson Education.

Van Der Aalst, W. M., Ter Hofstede, A. H., & Weske, M. (2003). Business process management: A survey. In *International conference on business process management* (pp. 1-12). Springer Berlin Heidelberg.