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

FLORIAN SCHMITT
STUDIENARBEIT

IMPROVING DOMAIN MODELING AND REQUIREMENTS ANALYSIS USING GROUNDED THEORY

Submitted on 8 June 2015

Supervisor: Andreas Kaufmann, M. Sc., Prof. Dr. Dirk Riehle, M.B.A.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander-Universität 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.

Erlangen, 8 June 2015

License

Erlangen, 8 June 2015

Abstract

One of the key factors of the success of professional software development is mature requirements engineering. This thesis focuses on the elicitation and analysis of requirements and addresses the process steps between the elicitation of requirements and the gathering of the system requirements specification and domain model.

We consider the state-of-the-art of these aspects of requirements engineering as suboptimal and propose an approach that implements the elicitation and analysis of requirements as an adaption of the Grounded Theory approach, which is methodically sound at the field of social studies.

Our approach will implicate the institution of an additional artifact within the process, the so-called code system. Furthermore this approach enables the direct mapping from the gathered information about the target domain, represented in the code system, into a domain model.

Contents

1 Introduction

1.1 Original Thesis Goals

This thesis aims at the development and realization of an approach that implements the elicitation and analysis of requirements as an adaption of the Grounded Theory approach, which is methodically sound at the field of social studies. The developed approach is applied to practical execution at the openETCS project.

Originally this thesis was intended to determine the requirements for the tool chain throughout the whole openETCS project through the exploratory application of the newly developed method based on Grounded Theory. The intended process steps were the execution of a series of interviews with project stakeholders, the processing of the associated interview transcriptions with the tool MaxQDA and the implementation of a code system that offers a direct mapping from statements from the analyzed interviews and documents into a domain model which is automatically deduct-able from the code system.

Furthermore, the code system was intended to feature a direct deduction of a glossary with the specialized terminology of the target domain.

1.2 Changes to Thesis Goals

As it turned out that the analysis of the entire project tool chain of openETCS would not be manageable within the time scope of this thesis, the scope of research was confined and we focused on the analysis of requirements engineering within openETCS and the associated tool chain.

In addition, the time restrictions forced us to resign the development of a tool that automates the modeling. However, all necessary data structures are given and the data representation which we developed for the domain modeling, which is stored within the memos, was tested during the manual mapping into a domain model, only the tool that features the automation of this mapping had to be shifted to further research.

We therefore built a domain model manually, according to the memo contents.

2 Research

2.1 Introduction

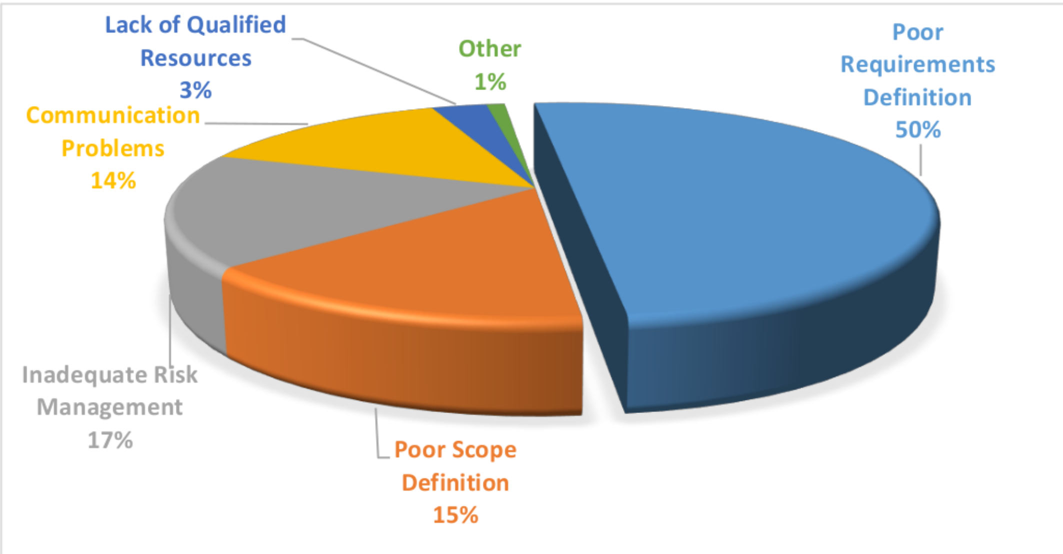

Figure 2.1: Why software projects fail (data from (ESI, n.d.))

The quality of requirements is one of the major factors of success of any software development nowadays (see figure 2.1). The requirements quality, however, is greatly influenced by the techniques employed during requirements elicitation (Hickey & Davis, 2003, p.169).

Requirements Engineering (RE) is one of the key disciplines of software development. It covers the elicitation, analysis, and specification of requirements plus the management throughout the development process, particularly including requirements change management, as can be seen in figure 2.2. This thesis focuses on the aspects of requirements elicitation and requirements analysis and neglects the other aspects.

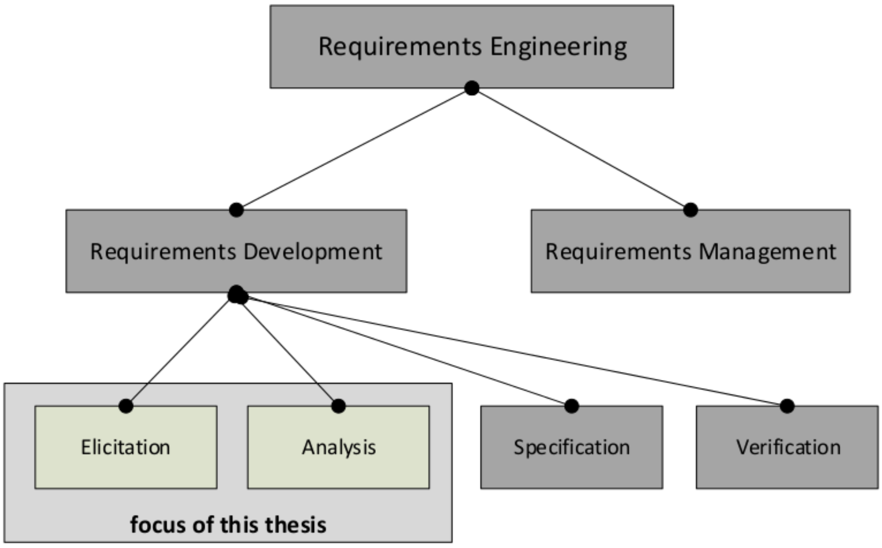

Figure 2.2: The subdisciplines of Requirements Engineering

Chapter *3.1 Requirements Analysis: The State of the Art* introduces the details of the process of requirements elicitation and analysis as it is currently done. It is shown that in many aspects the analysis and the decisions which the requirements analyst takes is done by intuition and with implicit knowledge, and that the process from the starting of elicitation up to the deriving of a SRS is insufficiently documented. Many sequences of operations are grounded more in practical experience of the analyst than in clearly structured decision guidelines.

We propose a new approach, based on the adaption of Grounded Theory (GT) for requirements analysis, which will implicate the institution of an additional artifact within the process, the so-called code system.

Typically the domain analysis includes the establishment of a so-called domain model (DM). DMs represent the key concepts and the specific vocabulary of the domain and aim at describing and constraining the scope of the domain. This approach enables the direct mapping from the gathered information about the target domain, which is represented in the code system, into a DM and therefore features substantial improvements concerning traceability, documentation, systematics, rigorousness and maintainability.

2.2 Related Work

2.2.1 Application of QDA Methods to Requirements Engineering

There has not been much research up to date that investigated the application of qualitative research methods to the elicitation of software requirements, but a few exceptions can be associated.

(Coleman & O'Connor, 2007) used the GT method to elicit systems and user requirements in the context of software process improvement for the Irish software product industry as a test-bed. They both described the usage of GT in the study and evaluated its effectiveness as a research methodology and concluded that GT is a reliable method for this purpose.

(Kaufmann & Riehle, 2015) investigated the potential of RE-methods based on Qualitative Data Analysis (QDA) to improve traceability and concluded that the processes of RE and theory building are similar and the adaption of QDA methods is a suitable method to improve the early development stages with respective to traceability and change management.

Halaweh proposed the use of GT as an alternative approach for requirements analysis in (Halaweh, 2012). He demonstrated the application technique through a case study and provided a mapping from GT outputs into a class diagram and states that GT can be used to improve the communication both between the analyst and the stakeholders and within the development team members by the easy transformation of GT models to standard models like UML. In addition he particularly addressed the capability of understanding users' needs better through the feasibility to address the nontechnical issues.

2.2.2 Domain Model Generation in Requirements Engineering

The state of the art of the process from Requirements Analysis (RA) over system modeling up to the development of high-quality requirement specifications and their representation in formal models like UML is presented very detailed in (van Lamsweerde, 2009). In particular, the critical role of the requirements engineer is discussed.

A further book which presents a detailed and high-qualitative overview of object-oriented analysis and design is (Booch, 2006), where sections focused abstraction levels can be found at pp.274-276 and pp.281-283.

2.3 Research Question

This research will address the following questions:

Is the use of a QDA-based approach an advantage for the processes of requirements elicitation and analysis?

Is the adaption of Grounded Theory suitable to improve the development of a domain model of the target domain?

Which advantages and disadvantages characterize the GT-adoption in comparison with the conventional approach?

2.4 Research Approach

To develop a new approach for domain modeling we applied traditional GT practices within an exploratory project. The purpose of this project was to understand the needs within the openETCS project towards their tool chain. In essence we performed requirements elicitation and analysis for the RE phase of the software development process within the openETCS project. The applicability of different qualitative research practices was evaluated towards their usefulness in gaining an understanding of the domain and their impact on the creation of a domain model.

One of the characteristics of our approach is that the elicitation of data is done iteratively. This particularly means that instead of collecting a huge amount of data in a first step and analyzing this afterwards in a second one, we collect and analyze data in parallel and can iteratively decide, which pieces of information to elicit next and where to find them. This proceeding is directly adopted from classic GT, where it is denominated as Theoretical Sampling (see also chapter *3.3.2.3 Theoretical Sampling*).

The iterative data elicitation is continued until no additional data can be found that further develops the categories of the domain to be researched. When all categories are saturated, the data elicitation is finished. This principle is defined in GT as Theoretical Saturation and closer described in section *3.3.2.7 Theoretical Saturation.*

2.4.1 Outline of One Iteration

The following section aims at introducing the reader to our research process by describing the steps of one iteration in detail. In application of the approach, these steps will be executed multiple times. Within our exploratory project, we conducted four iterations with three stakeholders from the openETCS project.

2.4.1.1 Execution of Interview

Typically, one of the main data sources when carrying out QDA is the execution of interviews, which provide several advantages in comparison to other data elicitation techniques: It is possible to individually fit the course of the interview and the researcher can directly ask questions at topics where he would like to go more into detail. Therefore the likeliness of misunderstandings and ambiguities is much lower than with i.e. the use of questionnaires.

2.4.1.1.1 Selection of Interviewee The most fundamental factor for a successful data elicitation by the use of interviews is the selection of the suitable interviewee.

Since the domain analysis process was done in an iterative way, we had to decide multiple times which persons to interview next. However, during the exercising our approach at the openETCS project, the concept of Theoretical Sampling could not be fully implemented. There were two reasons for this:

- Within this thesis, the time on hand to execute the approach was very limited. Usually, the interviewing of more people would be appropriate to reach Theoretical Saturation. Unfortunately, only four interviews could be carried out. This was sufficient to prove that the approach and its concepts are basically well-suited for the intended field of application, but more interviews would have further improved the quality of the designated DM.
- During the analysis process concerning the openETCS requirements engineering domain, it turned out that the number of available interview partners was commensurately low, since some of the potential interviewees had already left the project again and were not available any more.

The details of the interview specifics of the openETCS analysis are closer described in section *2.5.1 Interviews*.

2.4.1.1.2 Preparation of Interview To ensure an efficient interview execution and results of good quality, an interview guideline was prepared in advance. Although it would principally also be possible to do interviews without a prepared outline, we strongly recommend it, since it turned out that this significantly improves both the time efficiency and the quality of the gathered information.

In two out of four cases, we mailed the interview outline to the interviewee before the interview. In both cases, the interviewees had asked for the questions, since they wanted to make sure that they could provide information to all requested aspects. Yet, we conclude that the provision of interview outlines in advance does neither affect the efficiency of the interview nor the quality of results, therefore it can be considered as nice to have, but not necessary.

The interview questions originated in the collected data up to that point of time, respectively:

- **Gaps within the current code system** were one source of questions for the next interview . When we determined open points, the lack of deeper information or untreated aspects within a certain topic during the analysis of current data, this triggered additional investigation by questions in the next interview.
- **Discrepancies within the code system** typically were originated in contradictions in the statements of different interviewees. Therefore, these aspects were detailed in upcoming interviews and more data was collected, which either confirmed or contrasted the present statements.
- **New aspects** In many cases, during the analysis of existing data, we discovered new topics and aspects of the target domain within the analysis which were not at all treated up to then. Hence, they were added to the investigation and treated in following interviews.

A special case was the initial interview, obviously. Here, the questions and investigated topics arose out of the initially analyzed official project documents, which were provided by our industry partner to enable us to get an overview of the project in general and the specific aspects of the scope of the research.

With hindsight, it can be stated that the gathered data from the first interview was relatively general and superficial and therefore did not contribute very much to a precise and object-orientated representation of the target domain. Rather, it served us to get familiar with interconnections and interdependencies throughout the project as a whole, which is the basis for a correct delineation of the research scope and the identification of all interfaces with the domain environment. Therefore, we consider the initial interview as not very fertile in terms of detailed analysis, but nevertheless as essential and crucial.

2.4.1.1.3 Interview The actual interviews were all carried out as telephone interviews, which were audio-recorded.

We used open questions to create a loose atmosphere and encouraged the interviewees to talk about what came to their mind and also change the topic if they wanted to. The prepared questions served more as an outline.

The intention was to let the interviewee speak freely, which is also transferred from the traditional GT techniques. This shall ease the discovery of topics that the analyst might not yet be aware of. However, when statements came up that the analyst did not understand or included unclear details, or when the current interviewee contradicted statements from earlier interviews, the analyst inquired these points and asked for more details.

2.4.1.1.4 Transcription of the Audio Record In a next step, the audio records were transformed into transcriptions.

In one case, this was done with the help of a foreign transcription service and the received document was checked for correctness and completeness. The other three interviews were transcribed by ourselves.

For the transcription, we used Winamp as one of the common media players in combination with the add-on Pacemaker, which enabled the deceleration of the audio record. For the text processing, Microsoft Word was used.

2.4.1.2 Analysis of the Transcript

The obtained interview transcript was analyzed in the next step.

As our approach is based on qualitative data analysis, this means the processing of the transcript with a so-called *CAQDAS*-software, which stands for "computer-assisted qualitative data analysis software". We used MaxQDA within our research, which is one of the standard tools on the market. MaxQDA is basically capable of working with Word-Documents as well as pdfs or video/audio files.

After the import of the word-document into our project, the transcription was analyzed by executing the coding process as it is described in chapter *3.3.2.1 Coding*.

We deviated from the standard GT-approach as follows:

- The most radical modification is the **modified use of memos**.

  In conventional GT, memos serve for the recording of the researcher's thoughts, like questions that come to his mind, ideas how to progress or whom to interview next, etc. (see also section *3.3.2.2 Memos*).

  Within our approach, we used the memos for two purposes, which are illustrated in figure 2.3:

The first one was the conventional role as described above. The use of memos proved to be a valuable help to get thinkings denoted without the necessity to do this very structured or in mature language. In addition, the direct interconnection from the memo to the coding and its placement within the code system plus the possibility to directly access the related text sections from the interviews was very helpful. The upper part of figure 2.3 provides an example for such a conventional memo.

The second purpose is that we used the memos as the interfaces to denote all informations which were necessary to derive a domain model and a glossary out of the code system. This key aspect of the approach will be described in detail in chapter *2.4.2.1 Information representation in the Memos*. An example can be found in the bottom part of figure 2.3.

- In contrast to the conventional approach, which primarily aims at the development of theories that describe and explain human behavior, social processes and patterns, our research was targeted on an **object-orientated description of the analyzed domain**.

  Nevertheless, one of the advantages of our approach is the possibility to take the mentioned social aspects into account in addition. The focus on object orientation does not require the neglect of social aspects, in fact, it is easily possible to involve different views.

  Therefore all social aspects interconnected to the domain can be explicitly included in the code system. Hence, this information does not get lost but is represented in the artifacts, which we consider a major benefit of our approach.

2.4.1.3 Code System Revision

After running through the coding process, we have a new version of our code system, which we now revise and check for quality.

- Discovered **discrepancies** trigger the further investigation of the related topic in the next interview
- **unification of double occurrences** of codes
- **consistency check of the hierarchical structure** of the code system

Hence, the code system is smoothed and corrected. In addition, the memos are updated afterwards. Arisen thoughts and questions are denoted and often serve as the basis for discussions within the next interview.

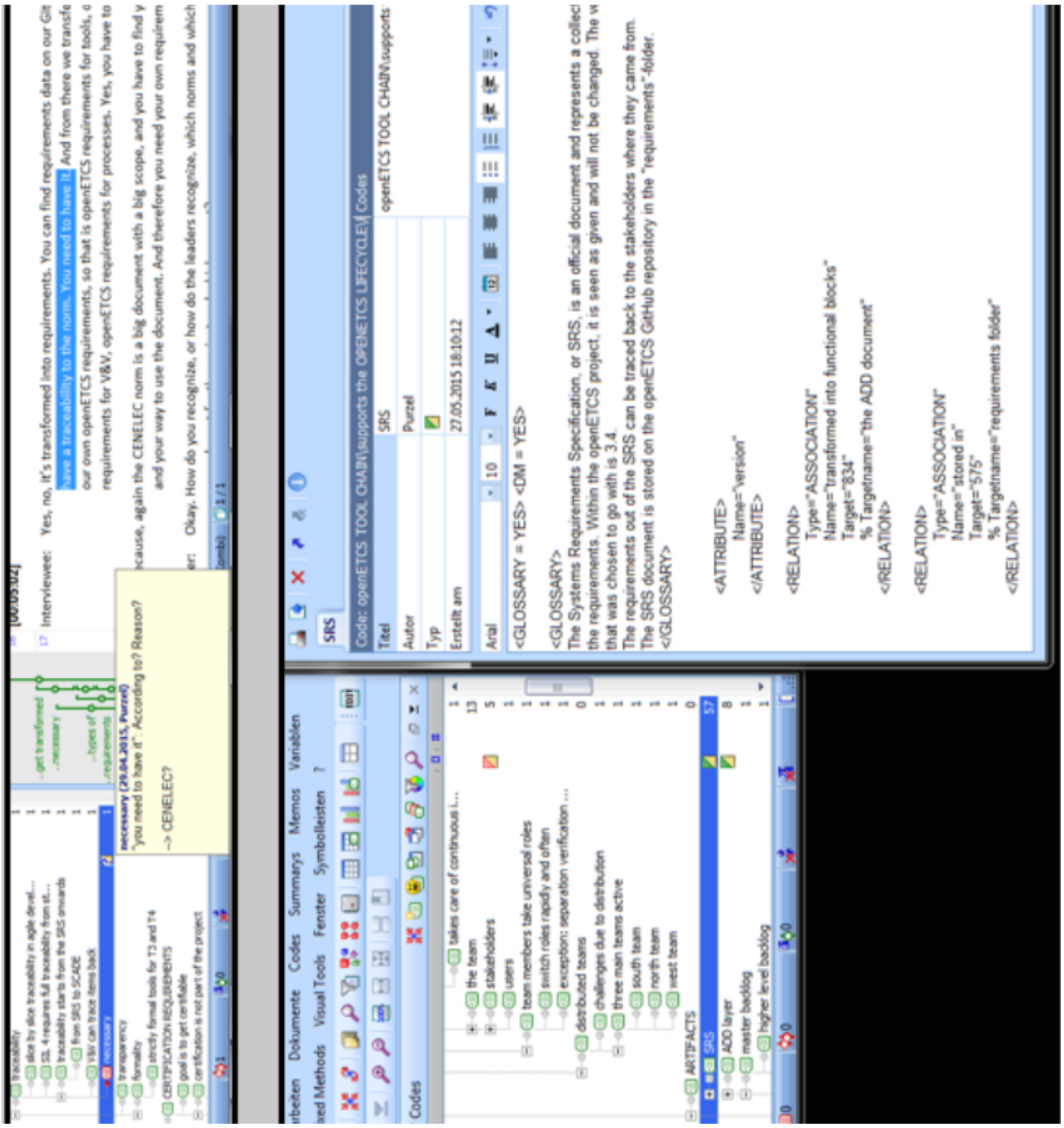

Figure 2.3: The two types of memos in comparison: Conventional (above) and new approach (beneath)

2.4.2 Extraction of Domain Model and Glossary

Chapter *2.4.1 Outline of One Iteration* described the process of elicitation of data. The following chapter will focus on the question how to embed this information within the code system in a way that allows to derive a DM and a glossary in an automated way.

Since the use of QDA for RE is rather new and the deriving of a DM out of a QDA code system has not been done before, of course there is no optimized software for our purposes. This might be the subject of future development, if research to come confirms the validity and suitability of our approach or related research. At present, we decided to go with MaxQDA, which is one of the market leaders of CAQDAS.

The features of MaxQDA are clearly suited for conventional QDA, but not optimized for our purposes. There is no designated area which is intended to serve for the embedding of additional information which outruns codings, a hierarchically structured code system and "conventional" memos, being used for the denotation of thoughts and questions. MaxQDA rather contains lots of powerful features which are suited for researchers from social sciences, like for example the detailed analysis of code frequencies or an interface to the statistics software SPSS.

We considered memos as the best option for us to encode the necessary additional information, since they provide a free field where it is possible to store bigger amounts of text. In addition, each text field is directly related to a code, which facilitates the mapping of an object-oriented structure a lot.

2.4.2.1 Information Representation in the Memos

Since memos can only store plain text, we needed to develop a structure to prepare the DM information in a way that makes an automated mapping to a DM possible.

Figure 2.4 gives an overview of the structure we developed. It consists of four sections, which were encapsulated with <XML >-brackets to facilitate the parsing.

1. **Two switches which decide if the code will be included in the glossary and in the DM**. This is necessary, since not all codes are relevant. Not all codes contain object-oriented information, in fact, the majority contains either meta-information (which is subsumed in the superordinate glossary-entry) or information which is irrelevant for the deduction of DM and glossary. However, these codes are kept in the code system and supplement the data which is collected to a concept.

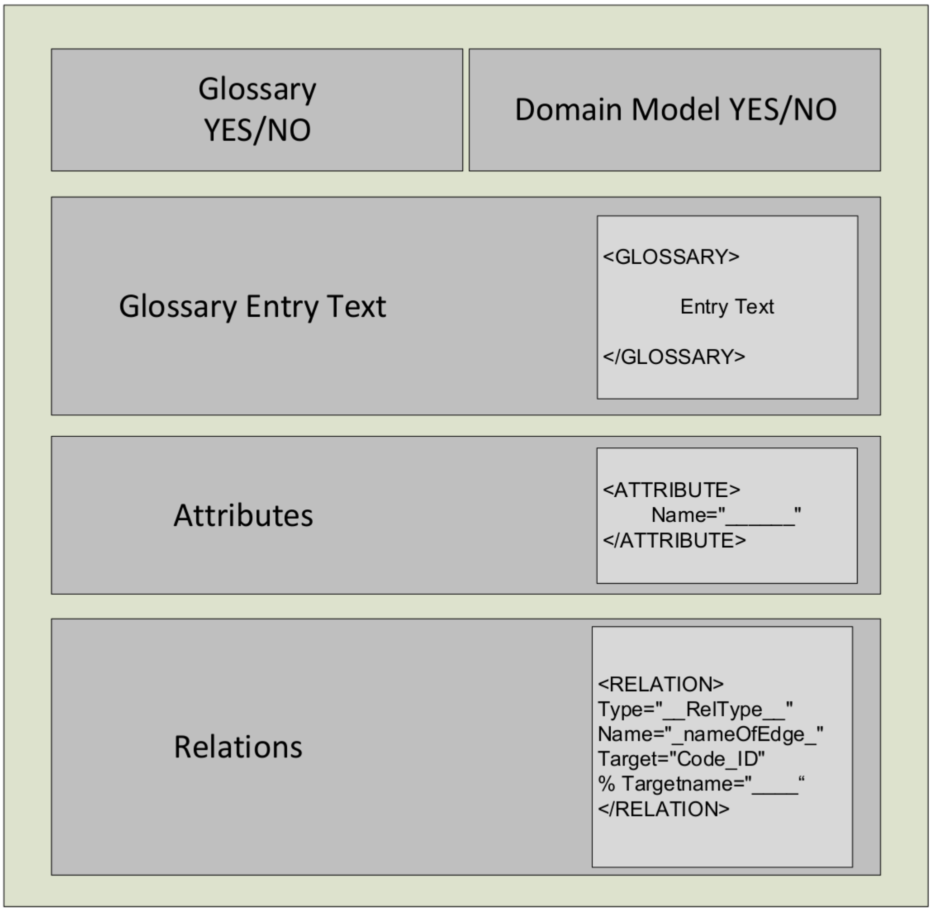

Figure 2.4: The Design of Memos for Domain Modeling

2. The **Glossary Section** includes a description of the concept. Here, the meta-information from the subordinate codes is also presented. If no glossary entry is wished, this section can be deleted and the glossary-switch be turned to NO.

3. The **attribute-section** contains the attributes of the code. Each attribute is stored as <ATTRIBUTE >Name=" "</ATTRIBUTE >. We decided that for this approach the only relevant information to be stored in an attribute is its name and that all additional attribute-values defined in UML (initial value, property value and assurances) will be ignored.

4. **The relations-section** is encapsulated again with <RELATION >and </RELATION >.

   It contains four entries:

   - **Type** defines the type of relation. Valid values were decided to be association, inheritance and realization. Generally, more types would be possible here, naturally. However, implementing some of them would have required more complicated structures. In addition, we decided that for a proof of concept these would be sufficient.

- **Name**: The name of the relation, respectively the inscription of the edge to the target code.
- **Target**: The code-ID of the target-code. This ID is taken from the exported XML file as described in section *2.4.2.2 The MaxQDA file system and the Mapping of the artifacts*.
- **Target-Name** includes the name of the target code. We decided to include it, although it is not used for the parsing, since the name of a code could be changed but its ID stays the same. This entry solely serves to facilitate the analysts work and is more a comment, which we want to emphasize with the "%"-sign.

We consider this structure a compromise between usability for the analyst and parse-ability for the software which maps the memos into the DM.

2.4.2.2 The MaxQDA File System and the Mapping of the Artifacts

Figure 2.5 shows the mapping process of the affected artifacts:

1. From MaxQDA, you can export the constituent parts of the MaxQDA project as XML files. What you get then is a folder which consists of three parts:
    - A sub-folder which includes all the documents which you imported into the MaxQDA project. The data format depends on the format of the files when they were imported: .rtf-files if you imported Microsoft Word documents or .pdfs if you imported pdfs.
    - A sub-folder with all memos from the code system in it, with one .rtf-file for each memo.
    - An XML-file, named like the project, which consists of three sections:
        - a "codings"-section for each imported document. It includes a description for each coding with the definition which text has been coded and its assignment to a code.
        - a "codesystem"-section. Each code definition includes an ID, the name of the code as a string, its color, the author who made the code and a time stamp when it was made.

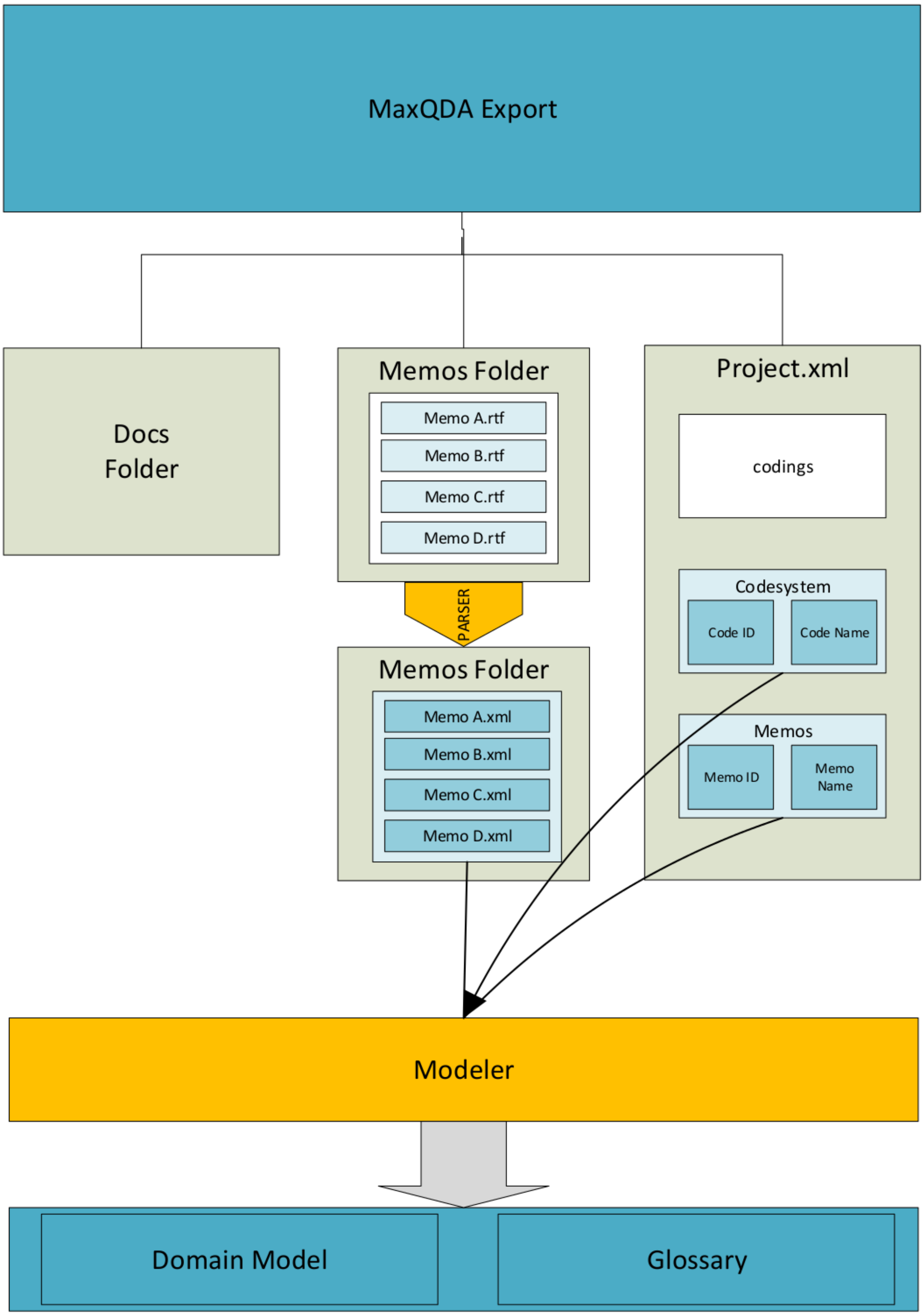

Figure 2.5: The mapping process from MaxQDA to the Domain Model

  - a ”memos”-section which defines each memo an ID, its title as a string, the author of the memo, a time stamp when it was made, and a relative path to the .rtf-file where the memo is saved.

2. The memos, which are stored as .rtf-files in MaxQDA, need to be transformed into XML-files, since .rtf-files cannot serve as input for a model generator. This can be done with the help of a tool that was already developed for related research, which can be found at `https://github.com/maclomork/qda-parser`

3. The relevant pieces of information (also highlighted in figure 2.5) are the memos from the memo-folder, and two types of relations, which can be extracted from the XML-file: the connection from code-ID to code, and the connection from a memo to the particular .rtf-file where it is stored.

   With the help of this information, it is feasible to derive a domain model.

   Unfortunately, it was not possible to program a modeling tool which implements this function within the scope of this thesis. Nevertheless, we believe this should be unproblematic.

2.5 Used Data Sources

Two major data sources were used for this thesis: We executed four interviews with three openETCS stakeholders and we processed the relevant official documents from the openETCS repository on GitHub and involved them into our coding process.

2.5.1 Interviews

The main source for collecting data within our approach was the execution of a series of interviews with stakeholders.

The people we interviewed are directly involved in the openETCS project as team members of the development team. Our interview series consisted of four interviews, which were all executed in a semi-structured way:

The initial interview took place in September 2013. Our interview partner was the project leader of openETCS. At this point, we had received an informal introduction into the project and the documents repository by a previous telephone conference with two of the project leaders and had already received an overview of the project.

In preparation for the first interview we analyzed the currently existing requirements document and carved out inconsistencies, imprecise wordings and mistakes. The discussion of these aspects served as an introduction to the different topics, but the interview became very open and often one aspect brought up the next one. This resulted in a proportionally long and detailed conversation which included the whole project.

The openETCS project is split up into work packages (WPs), with each of them focusing on one aspect of the development.

The second interview was executed in April 2015. The interviewee was the WP leader of work package 3, which concentrates on modeling. As this WP is the main user of the results of requirements engineering and therefore is a major stakeholder concerning the requirements engineering within openETCS, we decided to talk to its leader.

The third interview was carried out in May 2015. The interviewee was once again the project leader. The reason for interviewing him again was that he also represents the product owner from the agile development process and therefore holds a key role from the RE point of view.

A fourth interview was executed in May 2015 with a team member that acts in various roles within the agile development process: Scrum master, architect, verifier and sometimes as a product owner. He can be seen more as a normal team member than a project leader and therefore we emphasized his view and were very interested in his daily routine and his experiences.

We would have liked to talk also to the experts who did the initial requirements analysis at the project start. However, this task was fulfilled by partners from the French railway company SNCF and the experts who did it had already withdrawn from the project. Unfortunately, it was not possible to arrange an interview with one of them.

2.5.2 Further Data Sources

One of the principles and major advantages of the Grounded Theory approach is that almost every kind of data can be involved and processed (see also chapter *3.3.2.6 All is Data*). In the case of our example project openETCS the most relevant source of information besides executing the interviews was the involvement of documents from the openETCS repository on GitHub.

We considered it useful to take these documents into account out of the following reasons:

They helped a lot at the beginning of the research to get an overview of the project in general: What are the project goals, what will be the expected challenges and difficulties, who executes this multinational project and why, etc. Often, the official project documents turned out to provide lots of information about the particular topic of interest in a very well-structured, concise way.

Thus, we had the opportunity to acquaint ourselves with a particular topic before talking about it with a stakeholder, which helped a lot to make the interviews as precise and well-tailored as possible. This was our goal when preparing interviews because of two major reasons: Firstly, as our interview partners were all either WP leaders, product owners or project leaders, their availability for interviews was limited. Therefore we wanted to fathom additional information sources. And secondly, the transcription of an interview record and the coding and analysis of the transcription is proportionally time consuming and laborious. Thus, we had an interest in being well-prepared before starting a particular interview and avoid wasting time with negligibilities.

In addition, in some cases it turned out also during or after the interview that a certain document was the best data source for a particular issue, since the issue's specifics were depicted in a very good way there, where it would have taken a lot of time and effort to receive a description of equal quality in an interview.

Sometimes, we were also explicitly directed to a certain document from stakeholders or interview partners.

Thus, often the gathering of information was distributed on getting general information from existing documents and discussing specifics and details with the interview partners. However, there were also lots of topics that were constituted by our interview partners which were not described in pre-existing documents. This affected especially the subject areas of the agile development process and the team collaboration. Topics that were described in detail in existing documents were, for example, the distinction of primary and secondary tool chain and the various tool candidates.

2.6 Research Results

2.6.1 Views and Perspectives

When we started analyzing the first interview, we very much took care to let the concepts and theories emerge out of the data and treat the data as unbiased as possible. We tried to avoid the perspective of a conventional analyst.

Nevertheless, it turned out that our code system structure more and more developed into a direction which might be considered to be the expected outcome of a conventional analysis as well; and in the end our code system was split into sections which treated the agile development process, the tools, and the artifacts. These categories emerged out of the codings and codes, it simply made sense to order the codes this way.

We consider the QDA-based approach to provide techniques to elicit all domain-related information in an extensive way. It proved to be well suited to process structural information as well as process descriptions or non-technical aspects. We assess our code system to provide a substantial analysis of all these aspects, however we see the structural information better elicited than the process aspect.

But we believe that you cannot deduct a general evaluation out of this, particularly with respective to the fact that it was not possible to reach theoretical saturation within the scope of this thesis (see also chapter *2.7 Results Discussion*). We think that this proportional strength of structural information is based on the fact that we focused the interviews on these aspects as well, since we tried to use the limited time as effective as possible and therefore preferred to i.e. ask about which artifacts are used by which roles of the team and which tools are used in the context, rather than let the interviewee tell us in detail what his daily routine looks like.

Therefore we don't think we can rate the approach on a substantial basis in this context. Further research will be needed to enable an empirical evaluation.

2.6.2 The Degree of Freedom in Coding

Conventional GT gives the researcher very high latitudes how to develop the codings (see also section *3.3.2.1 Coding*). This section discusses whether this high degree of freedom has to be restricted for our approach to work.

We believe that there is no necessity to limit the latitudes of the analyst. Rather, we consider the method to work as intended only if the degree of freedom is kept sufficiently high, since this is the basis for the capability of GT to systematically denote all types of aspects into the additional artifact code-system, also the "soft" and non-technical ones, which we consider one of the major benefits of our approach.

This however implies the question how to map pieces of information into the DM.

It will have to be distinguished between codes which are relevant for the DM, and codes which are not. The latter can and should be further kept within the code system, since they provide meta-information which is also valuable and might prove to be useful during further analysis.

The parts which are to be mapped will be those which define classes, objects, functions and attributes. It will also depend from the context and from the desired type of domain model, which codes this will be, since it will differ how to transform the code system, as domain representation, into a class diagram or a state diagramm.

Other codes, which i.e. describe the reason of decision, will not be directly represented in the DM; however, they will be denoted and are easily accessible for the analyst when he researches this particular topic again, since he will find them directly interrelated within a logically and hierarchically ordered data collection of the domain.

The assignment to these categories of codes will have to be done by the analyst, with the use of implicit knowledge of the overall domain and the skill to know how to map the gathered information into the DM. From our point of view, this can not be automated or regulated, it requires professional experience and human intelligence.

Thus, the gap that we criticized, which arises through the huge amount of steps that are done by a conventional analyst with intuition, implicit knowledge and insufficient documentation, will not disappear completely.

However, we consider our approach as a substantial improvement, since it enables the extensive diminution of this gap. The institution of the code system as an additional artifact features a significant increase in traceability, maintainability and transparency, as it directly interconnects the statements from the elicitation sources to the concepts in the DM.

2.6.3 Abstraction Levels

Our method is capable to process pieces of information on all levels of abstraction, since it facilitates their hierarchical and logical ordering. An abstract concept will be found on a high level within the code system and its details will be subsumed in the subordinate levels.

It turned out that in general, the codes that were mapped into concepts of the domain model could typically be found on the middle levels. The highest code system levels provided an abstract perspective split and therefore a structural order, like i.e. "tools" vs. "artifacts", whereas the low-level codes mostly represented details of a concept.

2.6.4 Shortcomings of MaxQDA

During the execution of our approach it turned out that the use of MaxQDA as the CAQDAS of our choice implied lots of problems and shortcomings, however it has to be clearly said that they were limited to the step of mapping to a domain model and glossary. The previous steps were possible without any noteworthy problems at all, and during the entire process up to the implementation of data within the memos, MaxQDA proved itself to be a valuable help.

However, in the step of transforming the elicited data into the memo structure, we faced several challenges:

- To establish an interconnection to another concept (like i.e. associations), you need to reference that concept with an identifier. Within MaxQDA, every code is definition includes inter alia an ID and the name of the code as a string (see also section *2.4.2.2 The MaxQDA file system and the Mapping of the Artifacts*). Since the name of a code is changeable (which absolutely makes sense for QDA), the only unique identifier is the code-ID. But this ID has to be picked over the detour of a XML export as described in section *2.4.2.2 The MaxQDA file system and the Mapping of the Artifacts*, since this ID is not specified anywhere within the tool's front-end.

  This makes the definition of interconnections very complicated, laborious and error-prone.

- It is not possible to open a memo as part of the user interface, but only in front of it. This makes the using once again rather circumstantial.

- It is not possible to define more than one memo per code. This would be very useful, since you could split up between conventional memo use and the DM-defining ones. Furthermore you could also define several memos for several aspects, i.e. one for the glossary entry and one for the DM interrelations and attributes.

- It is not possible to mark the status of a memo, which would help for keeping an overview which DM memos have to be finished. Within this approach, we used a traffic light color-code as a workaround, but this was only a compromise; the status of a memo was coded with red (instanced but not implemented) to green (finished) plus the color blue for empty memos which were not relevant but helped to keep an overview within the code system.

- In some cases, you need to keep a concept at multiple places within the code system. For example, the concept "stakeholder" will be itemized within the section of the team members within the agile development process, but also as one of the sources for requirements. It would be possible to unify these codes (in this case this might be within the team member section), but this will substantially prohibit any overview in the requirements sources section then.

Such situations require the possibility to link codes/concepts or use a concept similar to pointers. Since this is currently not implemented, we had to define some concepts more than once. However, these instances have to be taken into account in parallel when you develop a glossary entry for the concept or when you model the interconnections. This inevitably leads to redundance and a high error rate.

- When entering the memo of a concept, you can only see the outgoing edges, but not the incoming ones, since they are embedded into the memo where they start to avoid redundance. This makes it hard to keep an overview.
- In general, it is extremely difficult to keep an overview within the memos. One reason is that you have to open each memo to see what is already deposited in it. In addition, you need to execute changes very manually. For example, to change the target of an edge in the domain model will mean to look up the ID of the new target in the XML file and to change it within the correspondent memo. It turned out that it is very hard to avoid mistakes here, especially when the DM gets bigger and more complex.

2.7 Results Discussion

This research is intended to serve as a proof of concept whether the use of GT is suitable for the analysis of a domain and the development of a domain model.

This thesis includes several shortcomings, which have to be taken into account:

- The number of interviews which could be carried out was very limited due to the scope of the thesis. With only four interviews, it was not possible to reach theoretical saturation. This can also be detected in the code system, which still includes gaps and unclear points.
- In addition, the pool of potential interviewees was also limited and therefore the selection of the next stakeholder to talk to was not possible as intended. Therefore the concept of theoretical sampling could not be satisfactorily applied.
- We consider the missing validation of our results as the major shortcoming. Unfortunately, there was no possibility to let our partners from the openETCS project validate the quality of our results, since they were not available at the final stage of this research. The reasons were an openETCS

project review which had to be passed, and that all our contact persons were on vacation afterwards.

However, we nevertheless believe that this thesis provides a proof of concept that the theoretical concepts of GT can be applied to RA and that the use of a QDA-based approach in domain analysis provides multiple benefits and therefore represents a suitable method to introduce several improvements to this process.

We consider the proportionally high operating expense of this approach as its major impediment. This is especially true when the researcher is not familiar with GT and its concepts. Therefore we think that this approach tends to be better suitable for bigger and more complex analysis projects, since then its concepts will accentuate the improvements in terms of systematics, traceability and efficiency.

It will be necessary that future research further investigates the suitability of our approach for RA. It was not possible to exhaustively cover all affected aspects within the scope of this thesis, and since the proposed methodology is new and rarely researched, further effort will have to prove the value of the proposed approach.

The most challenges we faced were grounded in the use of MaxQDA as execution tool for our approach. However, MaxQDA cannot be blamed for this, since it was never developed for the methodology we used it for.

In conclusion, we think that a systematic, high-qualitative and efficient analysis would be feasible with an adequate tool. The development of such a tool, that brings QDA-based elicitation methods and efficient, well-usable modeling features together, will be part of our future research.

2.8 Conclusions

This thesis proposes a new approach for the elicitation and analysis of data in the field of requirements engineering, which capable to be used with all kinds of data sources that are based on natural language, like interview transcripts, audio files, norms or official publications. It adopts the Grounded Theory approach, which is well established in the field of social sciences.

We show that by the use of this approach, it is feasible to realize a higher quality in terms of traceability, systematics, maintainability. This is reached by the institution of an additional artifact, which directly relates the statements from the elicitation sources to the concepts in the Domain Model.

3 Elaboration of Research

3.1 Requirements Analysis: The State of the Art

3.1.1 Placement of Requirements Analysis within the Requirements Engineering Process

In every software development project the requirements engineering is highly dependent on the project's specifics and there is no standardized, unified process which can serve as a template. However, there are certain patterns and best practices that are valid for almost all development processes.

Figure 3.1 provides an overview of the Requirements Engineering process. According to (Sommerville, 2011, p.37), four main activities can be distinguished in the RE process:

1. ***Feasibility study*** An estimate is made of whether the identified user needs may be satisfied, considering budgetary constraints and cost-effectiveness. The result should inform the decision of whether or not to go ahead with a more detailed analysis.

2. ***Requirements elicitation and analysis*** The system requirements are derived through analyzing existing systems, talking to stakeholders and potential users etc. This may involve the development of one or more system models, which help the analyst to understand the system to be specified. This thesis is focused on these activities.

3. ***Requirements specification*** The activity of translating the gathered information from the analysis activity into a document which defines a set of requirements. Two types of requirements are distinguished between: User requirements and system requirements (see chapter *3.1.3.1 Abstraction Levels*).

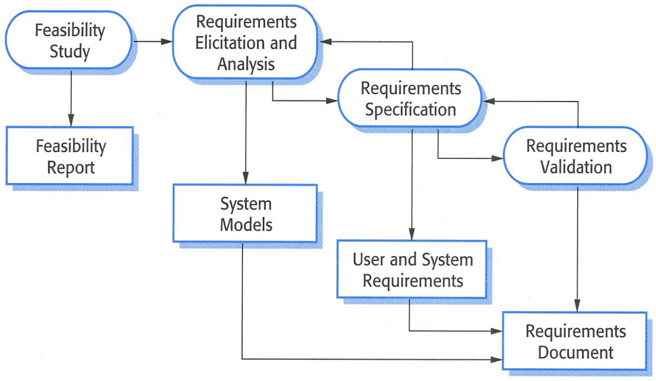

Figure 3.1: The Requirements Engineering Process (from: (Sommerville, 2011, p.38)).

4. ***Requirements validation*** This activity is about checking the requirements for realism, consistency and completeness. During this phase, errors in the requirements document are most likely discovered and must be corrected.

According to (Dumke, 2001, p.33), these activities, together with the requirements management, constitute Requirements Engineering, which is defined as the "systematic use of proven principles, techniques, languages, and tools for the cost-effective analysis, documentation, and ongoing evolution of user needs and the specification of the external behavior of a system to satisfy those user needs" (Marciniak, 1994, p.1043).

The described phases are not carried out strictly sequentially, but in an iterative way (see figure 3.2): Requirements analysis continues during the definition and specification phases and it is likely that new requirements will come to light throughout the process. Therefore, usually the first three phases from above are interleaved.

Development teams that use agile methods typically develop the requirements incrementally (from: (Sommerville, 2011, pp.37-38)).

3.1.2 Involved People

3.1.2.1 Stakeholders

Before the elicitation of requirements can be started with, it has to be clarified who sets the requirements. It will not be sufficient to survey the orderer, who pays the project (Balzert, 2009, p.504), since mostly the people who pay for the

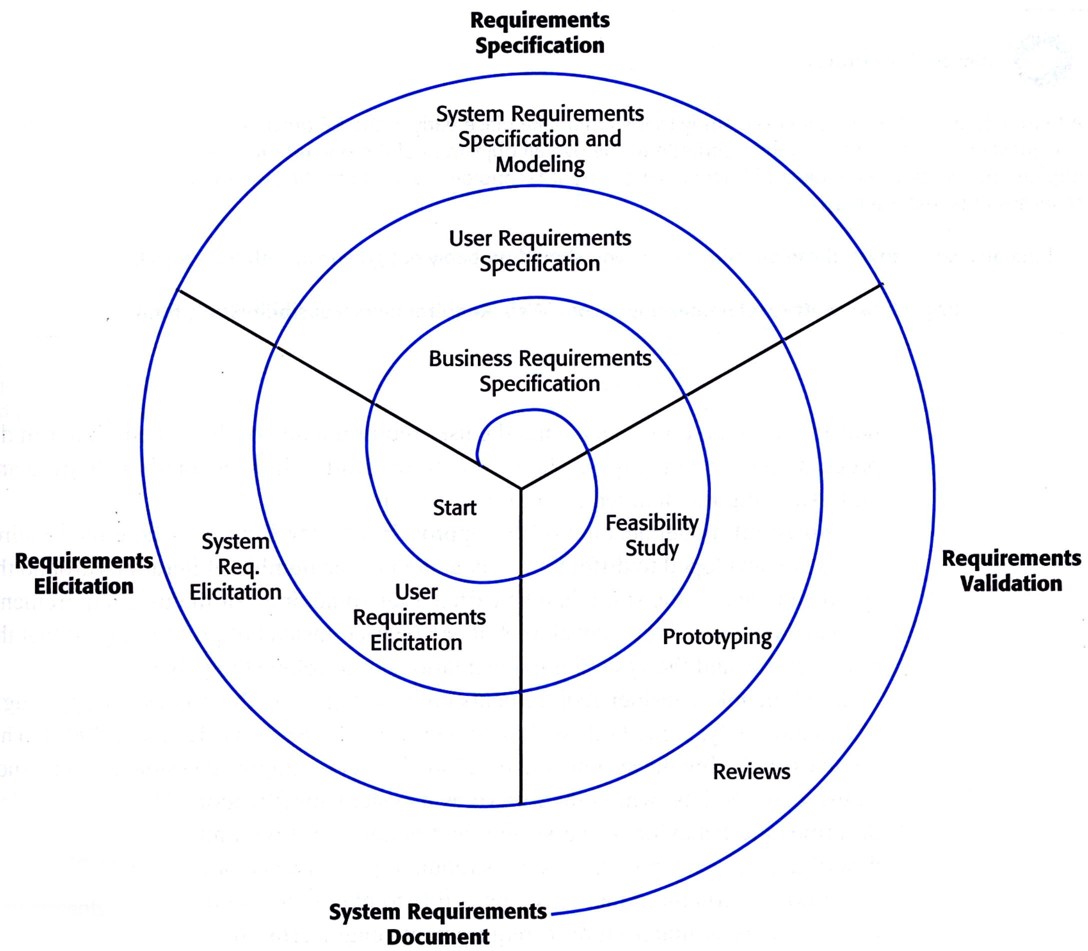

Figure 3.2: A Spiral view of the Requirements Engineering process (from: (Sommerville, 2011, p.99))

application are different from those who will be using it (Braude & Bernstein, 2011, p.231).

Therefore the first task must be to identify the stakeholders. This is often a step on its own within the process of requirements analysis. Moreover, many software projects develop a stakeholder-management-strategy explicitly.

In this context, we consider our approach to feature several benefits, since with our elicitation technique, the problem of whom to talk to is implicitly controlled by the concept of theoretical sampling. Therefore the identification of relevant stakeholders is easier. In addition, we expect a higher decision quality as well, since the decisions are iteratively taken with inclusion of all current data and the decision reasons are implicitly documented.
A **stakeholder** is anybody who should have some direct or indirect influence

on the system requirements, also anyone who influences the development, delivery and operation of the software product, including end-users who will interact with the system as well as engineers, business managers and domain experts ((Sommerville, 2011, p.101), (Balzert, 2009, p.455, p.504)).
The intensity that stakeholders influence the product with differs from stakeholder to stakeholder, both in a positive and a negative manner. It may be helpful to order and rate stakeholders concerning their relation to the product (positive, neutral or negative)and their corporate power. Figure 3.3 proposes how this can be done: Stakeholders are ordered by their influence and their conflict potential. Those, who are assigned to the upper right sector, permanently need to be monitored by the project management. If possible, it is advisable to influence these stakeholders positively.

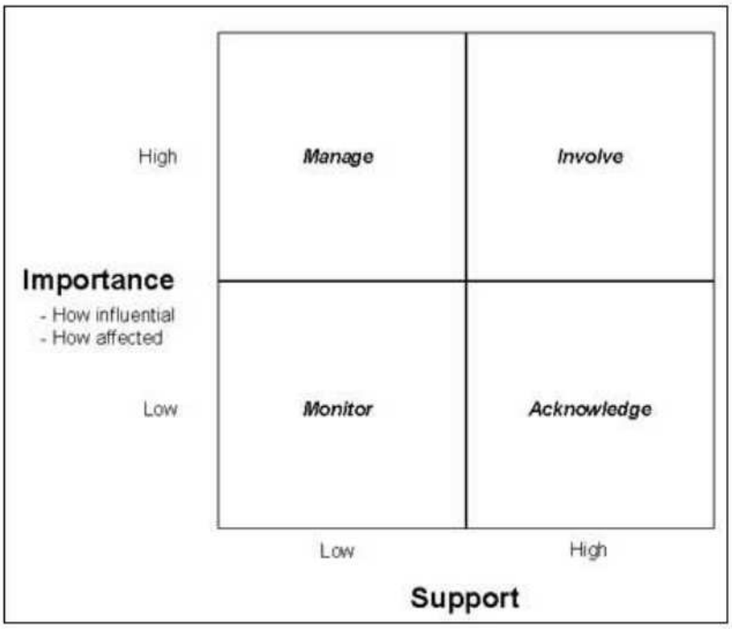

Figure 3.3: Stakeholderportfolio (from: (Prime Minister's Strategy Unit, 2004, p.79))

At the start of the requirement analysis phase, typically it will not be possible to identify all the stakeholders. Hence, it is necessary to pay attention to potentially additional stakeholders throughout the development process (Balzert, 2009, p.504f). As this is implied in our approach anyway, we consider it to be better suited to the elicitation process than the conventional stakeholder identification technique.

3.1.2.2 Project Environment

Each software system is embedded in a material and immaterial environment and this system environment has substantial influence on the requirements of the system (Balzert, 2009, p.461). Therefore it is important to define the system environment and especially its boundaries to the system.

The **project environment** is "the environment, in which a project is developed and realized, which influences the project and which is influenced by the project" (DIN, 2009-01)

This will typically be done in parallel to the discovery of stakeholders. This will include norms, standards and related laws. Additionally, ecological, economic, social, cultural factors may influence the project. These must be researched, collected, commented on and stored.

The system to be developed holds a **system boundary** that confines it from those parts of the environment that are not changed by the development. In this context, data sources and sinks need to be introduced. These are interacting with the system by the use of user interfaces and software interfaces. Thereby, data sources provide inputs and data sinks provide outputs. The interaction of the system with its environment is exclusively performed by using interfaces. Examples for sources and sinks are sensors, actuators, persons or other systems (Balzert, 2009, p.462).

Furthermore there is an environment around the system that is relevant and needs to be considered during developing the system. This environment is called **system context**. Its definition is fundamental, as it influences the way that requirements will be interpreted (Balzert, 2009, p.462). An example for the influence of the system context may be the bandwidth and stability of the internet connection of the device to be developed in context to time constraints.

3.1.3 Requirements and Types of Requirements

The requirements for a system are the descriptions of what the system should do – the services that it provides and the constraints on its operation (Balzert, 2009, p.455). These requirements reflect the needs of customers for a system that serves a certain purpose such as controlling a device, placing an order, or finding information. The process of finding out, analyzing, documenting and checking these services and constraints is called Requirements Engineering (RE) (Sommerville, 2011, p.83).

A requirement is (IEEE, n.d., p.62)

1. A condition or capability needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
3. A documented representation of a condition or capability as in (1) or (2).

Requirements can and should be classified in categories, considering the following aspects:

- levels of abstraction
- functional and non-functional requirements

3.1.3.1 Abstraction Levels

It is crucial to make a consequent separation between different abstraction levels of description. In the software industry, the term "requirement" describes both a high-level, abstract statement of a service or constraint of a certain system and a detailed, formal definition of a system function (and also every level in between) (Sommerville, 2011, p.83).

It is useful and necessary to state requirements at different levels of detail because they are used in different ways by different users. The same requirement on different levels gives information about the system to different types of readers: The typical reader of user requirements will not be interested in the details of the system's implementation. However, a person which needs to know about e.g. details of the system's software architecture will find these in the much more detailed SRS (Ludewig & Lichter, 2013, p.375f.).

We consider the code system of our approach to advantage this distinction, as the pieces of information are explicitly brought to a hierarchical order, where the superordinate code normally is more abstract and the subordinate codes provide details, meta-information and concretion. We expect this hierarchy within our code system to support the establishment of an abstraction level hierarchy within the requirements.

At the final stage, in the SRS, requirements are usually presented on two levels of detail: A high-level statement of abstraction, which serves for managers and customers, and a more detailed system specification which is needed by the system developers (Sommerville, 2011, p. 37).

Hence, a SRS is typically split into two parts (Braude & Bernstein, 2011, p.232):

- In the *first part*, high-level requirements are presented. This part is designed to be better readable and shorter than the very precise and detailed second part of the SRS. A stakeholder who wants to get an idea about what the project is about can read the high-level-requirements.

  In addition, this part often includes a project rationale.

- The *second part* provides a much more detailed and precise description of the requirements. This part aims at being used by system designers and implementers, who need to work with requirements which are elaborated on a very low level of abstraction.

More details about the SRS can be found in chapter *3.1.5.3 The Software Requirements Document (SRS).*

Sommerville (Sommerville, 2011, p.83) names the high-level, abstract requirements 'user requirements' and the detailed description of what the system should do 'system requirements'. In this thesis, we will adopt this nomenclature:

- ***user requirements*** are statements of what services the system is expected to provide to system users and the constraints under which it must operate. They are expressed in natural language, often expanded with diagrams.

- ***system requirements*** are more detailed descriptions of the software system's functions, services and operational constraints. They aim at defining exactly what is to be implemented. The system requirements are merged in the SRS, which may be part of the contract between the system buyer and the software developers.

3.1.3.2 Functional vs. Non-Functional Requirements

Requirements can be classified as either functional or non-functional requirements:

1. **Functional requirements**

   Functional requirements describe what the system should do: Which services it should provide, how it should behave in particular situations, how to react to particular inputs. Sometimes it can also be reasonable to state explicitly what the system should not do.

   Functional requirements may be written on different levels of detail and can be both user requirements and system requirements. When comprising a user requirement, usually functional requirements are described in an abstract way to ensure that system users can understand them. Yet,

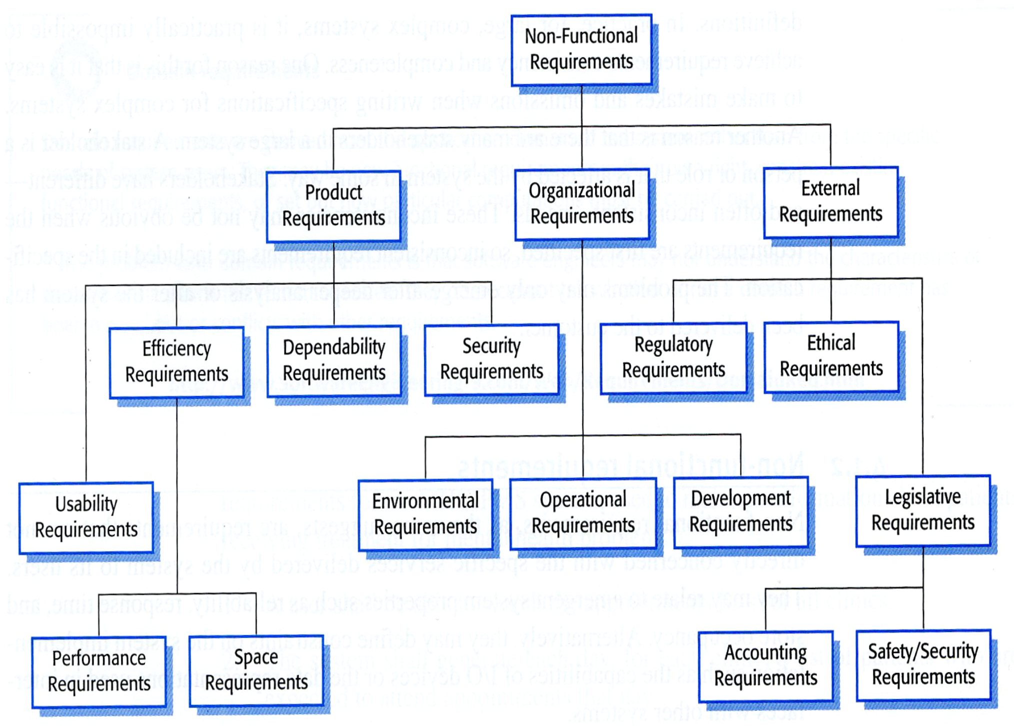

Figure 3.4: An overview of types of non-functional requirements (from: (Sommerville, 2011, p.88))

functional system requirements can also describe system properties in detail, like inputs, outputs, functions, exceptions, etc. (Sommerville, 2011, pp.84-87).

Following (Balzert, 2009, p.456), functional requirements can be divided into:

- requirements that define the **statics** of the system
- requirements that define the **dynamics** of the system
- requirements that define the **logic** of the system

The analyst should aim at making the collection of functional requirements as complete and as consistent as possible. In practice, it will be extremely difficult and time-consuming to achieve total completeness and consistency, since the avoidance of making mistakes in an environment with a complex system to be developed and with lots of stakeholders who have different and inconsistent needs is almost impossible. Furthermore, the discovery of inconsistencies itself is not trivial, since these inconsistencies are often not obvious but come up in later development stages (Sommerville, 2011, p.87).

Considering this aspect, we believe that the establishment of a code system as additional artifact plus the iterative character of our approach will promote a faster and better recognition of gaps and discrepancies (see also chapter *2.4.1.2 Analysis of the Transcript*).

2. **Non-functional requirements**

   Non-functional requirements do not directly deal with services or functions of the system to be developed. Typically, non-functional requirements concern the system as a whole rather than individual system features or services. They define characteristics of the system as a whole and often substancially influence the software architecture (Balzert, 2009, p.463).

   They relate to system properties like stability, reliability, accuracy, availability, safety, usability, or response times (list not complete) (Ambler, 2008, p.64).

   They can be founded in external factors like safety regulations and compliance to standards or laws or arise through internal constraints, like budget constraints, the need for interoperability with existing software or constraints on the development process.

   Often non-functional requirements are more critical than functional ones: If a system function contains particular shortcomings and does not meet the needs of the system user, he will find a way to work around. But to fail a non-functional requirement can put the whole system into question (Sommerville, 2011, p.87). For example, if the photo application of a mobile phone does not feature a filter function for the taken photos, this may annoy the user, but if the usability of the main menu fails in terms of its interaction with the touch display, this will put the whole device into question. Another example from the context of the openETCS project might be an on-board unit that implies severe shortcomings in terms of safety. In a highly safety-critical environment such as openETCS, this will make the whole product unusable.

   Figure 3.4 provides an overview of non-functional requirement types. The three main sources of non-functional requirements are (Sommerville, 2011, p.88):

   (a) **Product requirements** define the behavior of the software and its characteristics. This includes i.e. reliability goals, usability, safety, security, hardware restrictions etc.

   (b) **Organizational requirements** come from the organization which develops the software. Mostly they contain process- or development environment standards to consider, further examples are operational process requirements that describe in what way the system will be used.

(c) **External requirements** include all those requirements that come from external factors to the development process of the system and the system itself. This includes mainly legislative regulations and cultural and ethical requirements which make sure that the system will be acceptable in its designated environment.

Where at functional requirements it is mostly possible to define which system components implement a particular requirement, this is substantially more difficult with non-functional requirements. Often the implementation of a non-functional requirement can not be related to specific components, but diffuses within the system. Reasons for this are essentially:

- Non-functional requirements are often more related to the overall system architecture than particular components.
- A non-functional requirement may imply a number of related functional requirements that involve new system services.

It has to be advised that the classification of different types of requirements is often not clear-cut in reality. Requirements which are clearly non-functional on user-requirement level will most likely lead to additional, functional requirements when they are developed in more detail. Requirements always have interconnections to others, and one particular requirement may generate or constrain another one (Sommerville, 2011, p.85)).

Since these interconnections are made explicit in a structured and systematic way at our approach and get visualized within the DM, we consider it to help with this complex of aspects and finally result in requirements of higher quality.

3.1.3.3 Acceptance Criteria

Stakeholders often express the desired requirements in a very general way. A well-known example is the statement "The software should have good usability, and the frotend should be nice, maybe a bit like Apple", which most requirement analysts will have heard at least once.

Every good requirement should always come with clear acceptance criteria. It must be clearly visible when a requirement is fulfilled. This is especially true for non-functional requirements (Rupp & SOPHISTen, 2014, p.275).

Acceptance criteria need to be testable. During later development phases, especially the testing of the software, there must be clear acceptance criteria, and it has to be possible to test the product against these.

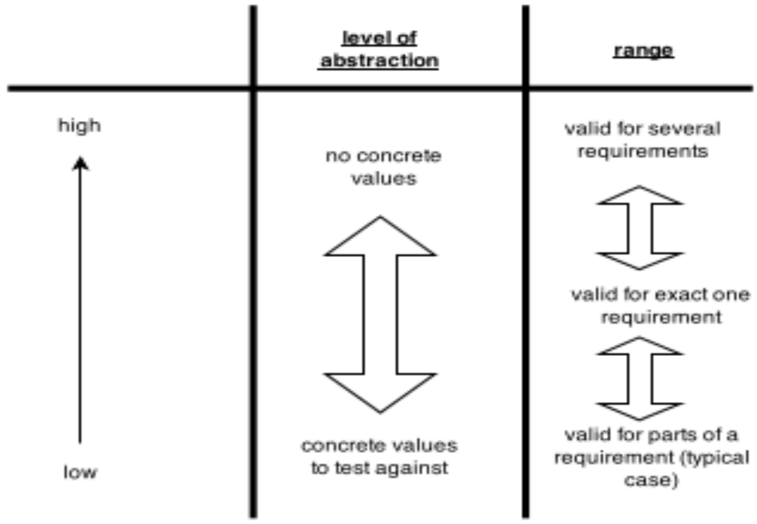

Figure 3.5: A classification of acceptance criteria in abstraction level and range (according to (Balzert, 2009, pp.471f.))

A requirement that does not contain clear, testable acceptance criteria is most likely not helpful and presumably can be taken out of the specification.

Acceptance criteria can be classified in level of detail and in their range, as shown in figure 3.5. The range of acceptance criteria can move within the validity for a part of a particular requirement up to the validity to several requirements. Considering the abstraction level, abstract acceptance criteria do not contain concrete values at all, whereas concrete acceptance criteria provide concrete values which can be used for i.e. testing against.

In addition, there is a number of **quality requirements to acceptance criteria** (Balzert, 2009, pp.471f.):

- Acceptance criteria need to be validate-able in an economic way, thus a test whether the requirement is realized needs to be possible with justifiable effort.
- Acceptance criteria needs to be testable on correctness, i.e. a testing if the requirement is realized correctly needs to be possible.
- Acceptance criteria need to be formulated in a way that allows to use them for regression tests.
- Acceptance criteria must not determine more or less than the related requirement demands and must not imply any additional properties or performance that are not defined in the requirement. Vice versa, details from

the requirement must not be omitted in the related acceptance criteria.

- Acceptance criteria should be minimal, but complete. They should cover all requirements, but not multiple times because of economic reasons.

Typically, stakeholders are no experts in requirement engineering and even when they are aware of the need of measurability, they find it difficult to translate their ideas into measurable requirements. Besides, they often can hardly estimate what a particular number as acceptance criteria means in terms of their everyday experience the system. Additionally, there are several non-functional requirements which can hardly or not at all be translated into some metrics.

We believe that the explicit denotation and the covering of the statements of multiple stakeholders will substantially assist the analyst to understand the stakeholders' needs.

In summary, it is very challenging for the analyst to transport the theoretical requirements of requirements and their acceptance criteria into the practical project. Sometimes the establishment of objectively measurable requirements is in fact possible, but very costly, and the customers who pay for the project may doubt the justification of that costs (Sommerville, 2011, p.90).

However, providing acceptance criteria for each requirement within the RA-phase implicates several advantages (Balzert, 2009, p.471):

- Requirements can be easily tested if being implemented correctly.
- Validation is not only grounded on the realized system.
- During formulation of the requirements it is already checked that they can be validated. This will lead to an improvement of the requirement quality.
- The formulation of acceptance criteria leads to a better illustration and understandability of the mostly abstract-formulated requirements.

3.1.4 Requirement Elicitation and Analysis

As described in chapter *3.1.1 Placement of Requirements Analysis within the Requirements Engineering Process*, the initial step of Requirements Engineering can be a feasibility study, which serves to assess if the system is useful to the business, to estimate costs and the impact of the system (Sommerville, 2011, p.99). The step to follow is the elicitation and analysis phase, which is the scope of our approach.

During this activity, the analysts work with customers and system end-users to find out about the application domain, what services the system should provide,

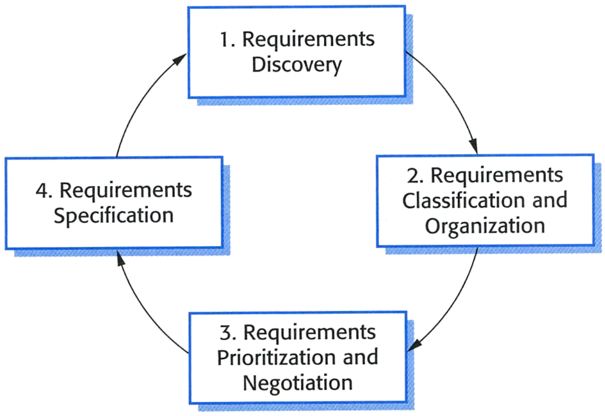

Figure 3.6: The requirements elicitation and analysis process (from: (Sommerville, 2011, p.101))

the required performance of the system, hardware constraints, etc. (Sommerville, 2011, p.99f.).

Generally, the requirements elicitation and analysis process is divided up into four main activities (Sommerville, 2011, p.101), which are all interconnected as can be seen in Figure 3.6. It shows that requirements elicitation and analysis is an iterative process with continual feedback from each activity to other activities. The analyst will understand the requirements better with each iteration.

The approach we propose is tailor-made for the nature of this process. This becomes very clear when figure 3.6 is put into context with chapter *2.4.1 Outline of One Iteration*.

3.1.4.1 Requirements Discovery

When the first stakeholders are identified and the project environment is evaluated, the first requirements can be determined and documented.

This phase aims at collecting as much information about the domain and the target system as possible. During this process, requirements will be formulated that describe the system to be developed in a manner that is as precise and as complete as possible: How the system should behave, how it interacts with its environment and in which context it is placed (for more details see chapter *3.1.3 Requirements and Types of Requirements*). The elicitation phase aims at making this collection of requirements as complete as possible, nevertheless in practice it will only produce a first version which will be improved in the further process.

Generally, elicitation techniques can be divided up into interviewing techniques and observance techniques. The approach which is proposed in this thesis is suited to process all types of results of these techniques, with potential shortcomings referring to the apprenticing technique. This will depend on the choice of the analyst how to denote the results.

3.1.4.1.1 Interviewing Techniques This is the most used category of elicitation techniques. Interviews are particularly suitable to retrieve the explicit knowledge of the stakeholders. They can be of two types (Balzert, 2009, p.507):

1. **closed interviews**, where the interviewees are asked a pre-defined set of questions. Often the analyst uses requirement templates, directing the stakeholders from visions and goals down to requesting acceptance criteria.
2. **open interviews** which are characterized through the lack of any predefined agenda. The analyst explores a range of issues together with the interviewee and hereby develops a better idea of their needs.

In practical elicitation work, interviews with stakeholders are mostly a mixture of both of these. Completely open interviews typically don't work well. The analyst will have to direct the stakeholder to focus the interview on the system to be developed. Typically, the analyst will ask certain questions to obtain answers to particular aspects, but when these lead to other issues he concedes room for the less structured discussion of those (Balzert, 2009, p.507).

Basically interviewing stakeholders is characterized by the possibility to individually fit the course of the interview, i.e. to answer inquiries, enumerate examples or go more into detail at unclear points. Usually the use of audio recording is helpful. The main drawback of interviews is the high costs in terms of time. Sometimes it might also be an option to interview several stakeholders at a time or to carry out workshops, but doing this efficiently requires lots of operating experience (see also (Rupp & SOPHISTen, 2014, p.106ff.)).

3.1.4.1.2 Observance Sometimes it may not be possible for stakeholders to participate in interview elicitations, i.e. because of lack of time, and sometimes it is very hard for stakeholders to express what is in their mind explicitly in words. In such situations, observing the stakeholder can be an appropriate way of requirement elicitation.

At observation techniques, stakeholders are observed by the requirements engineer while proceeding business processes. The stakeholder can be passive (and is just observed) or active (and explains what he does to the analyst) during that process. The analyst recognizes inefficient processes and proposes improvements. Two observance techniques are often used (Rupp & SOPHISTen, 2014, p.103ff.):

- **field observation**: The work of the system user is monitored in terms of activities and time lapses.
- **apprenticing**: The analyst is instructed by the user and learns to do his work. This technique is also suitable to get an idea about work sequences which are difficult to observe.

3.1.4.1.3 Further Elicitation Techniques

Further questioning techniques are:

- handing out **questionnaires** to the stakeholders (printed or digital)
- letting the stakeholders who are using the current system **write down memorandums** which describe their activities and sequences of operations plus their requests to the new system
- **on-site customers** who are constantly available by working directly with the development team. This is typical for Agile Development.

3.1.4.1.4 When to Use which Technique

It became clear that the elicitation of requirements is performed in a wide variety of situations, which depend on participants, the target domain and organizational contexts. In addition, it is also done with many instruments, which were described in overview in this chapter.

Many publications can be found that describe *one* way to perform requirements elicitation. However, there is not the one silver bullet solution that works best for all situations (Hickey & Davis, 2003). Consequently, almost all general requirements books describe multiple requirements elicitation techniques, like i.e. ((Balzert, 2009), (Braude & Bernstein, 2011), (Rupp & SOPHISTen, 2014) or (Sommerville, 2012)).

However, there are neither clear rules when to use which elicitation technique nor explicit favorites from experience. The specifics of the project indicate the appropriate ones in each case.

(Hickey & Davis, 2003) states that "less experienced analysts often select a technique based on one of two reasons: (a) it is the only one they know, or (b) they think that a technique that worked well last time must surely be appropriate this time".

She further emphasizes that the more experienced an analyst is, the more successful he uncovers the user needs. Consequently, she researched the method choice reasons of "some of the world's most experienced analysts" (Hickey & Davis, 2003) to find out if there are patterns or trends. She resulted that some general trends can be denoted, but nevertheless different very experienced experts choose different elicitation techniques when asked to analyze the exactly same problem domain(Hickey & Davis, 2003).

3.1.4.2 Requirements Analysis, Classification and Organization

During this activity, the unstructured collection of requirements is taken and organized by grouping related requirements together. Typically the requirements are grouped by using a model of the system architecture to identify sub-systems and to associate requirements with each sub-system. Hence, it becomes visible that requirements engineering and system design activities are interrelated and cannot be completely separated (Sommerville, 2011, p.101).

3.1.4.3 Requirements Prioritization and Negotiation

This activity is concerned with finding and resolving requirement conflicts that the list of requirements will most likely include, since multiple stakeholders are involved who have different requirements (see also chapter *3.1.4.5 Challenges and Difficulties of Requirements Elicitation*).

The finding of these conflicts is one of the core features of the approach we propose. Our process aims at doing this implicitly, and as early in the analysis process and as systematic as possible.

Typically, the resolution of this conflicts is done by prioritizing requirements and finding compromises with the conflicting stakeholders (Sommerville, 2011, p.101). This will mean that those stakeholders who defined the conflicting requirements come together and negotiate their importance and which opinion will become accepted.

3.1.4.4 Requirements Specification

In this phase, the requirements are transformed into well-ordered, structured and documented lists of requirements. These documents may be formal or informal. The elicited requirements are documented in a way that makes them helpful for the discovery of new requirements. Additionally, it is possible to generate an early version of the system requirements specification document (SRS) with missing sections and incomplete requirements (Sommerville, 2011, p.102). More details about the SRS itself can be found in section *3.1.5.3 The Software Requirements Document (SRS).*

3.1.4.5 Challenges and Difficulties of Requirements Elicitation

It is often difficult for the analyst to discover and understand requirements from system stakeholders for various reasons:

1. Stakeholders often find it difficult to articulate what they want the system to do. Unrealistic demands may be expressed, since stakeholders are often not capable of estimating which features are feasible and which not.
2. Naturally stakeholders express requirements in their own terms and with implicit knowledge of their own work. An analyst, without the same level of experience of domain knowledge, my not understand these requirements.
3. Different stakeholders have different requirements, and they may express these in different ways. Therefore it is crucial for the analyst to find all potential sources of requirements and precisely inquire them for commonalities and conflicts.
4. The analyst will have to consider political factors. Stakeholders for example may demand requirements because these will allow them to increase their influence.
5. The analysis takes place in a dynamically changing economic and business environment which will change during the analysis process. As a result the prioritization of requirements may change and new stakeholders may come up with new requirements.
6. Technical frame conditions may be ambiguous or even unknown during the RA phase.

Since different stakeholders will have different views on the priorities of requirements and these views often collide. Hence, one of the major tasks of the analyst is to bring the stakeholders together and negotiate compromises. Often, it will not be possible to satisfy every stakeholder, but the analyst needs to avoid that stakeholders start to undermine the requirements engineering process because they feel like their view is not being taken into account sufficiently (Sommerville, 2011, p.102).

Considering this aspect, the proposed approach provides additional benefits, since the methodology is particularly suitable to register the stakeholders' opinion not only to specific domain-items, but also to more abstract and soft factors like i.e. social aspects or the business policy of a company. While such topics might be omitted within a conventional analysis, they will be recorded and also represented in the code system with our approach and therefore are still present when they might turn out to be helpful at a later point of time.

3.1.5 Results and Products

3.1.5.1 Description of the Project Environment

The discovery of the project environment should result in a list of elements of the project environment. Often, this information is represented using mind maps. They are particularly suited to represent these elements, since it is possible to collect and structure information by the use of main and subordinate branches, whereas it is possible to use colors, symbols, different fonts and also include pictures. Hence, they enable both a structured representation and a fast and easy way to model loose pieces of information. As a result, mind maps provide an overview over a particular topic and its aspects and therefore inspire new ideas (Balzert, 2009, p.506).

The representation of such kind of data is one of the strong points of our approach. The code system includes all the information anyway, and since it was processed and assessed by the analyst, it is already structured and correlated to the other entities. At a point of time where the analyst decides to prepare such a description of the project environment, he has already collected relevant information, which is deposited within the code system if our approach is carried out. Therefore the description of the project environment is already given there, and the approach features the deduction of a domain model anyway.

If the analyst wants to adapt the representation or i.e. limit the information which shall be represented in the domain model, this can easily be done by customizing the mapping process from the code system into a graphic representation. It is possible to make a determination which types of information to take into account here, and it is also possible to save this determination as a template.

Moreover, the updating of the representation is featured implicitly, since the mapping of the code system into a graphic representation is possible in a very easy way, once the mapping is initially defined. Hence, our approach features several benefits and reduces the analyst's work at the same time.

3.1.5.2 Results of the Elicitation Phase

The requirements elicitation phase aims at gathering as much information as possible about the system to be developed and its domain and generating requirements out of this information. Requirements describe what the system should do and the constraints on its operation. The goal of the elicitation phase is to generate a collection of these requirements which is as complete as possible.

After the initial requirements elicitation phase, the requirements will not be very detailed, especially at the first iterations of elicitation, and in addition they will not be represented in a very formal way (Rupp & SOPHISTen, 2014, pp.37ff.).

Hence, the collection does not necessarily have to be a list written in prose, similar to a SRS. Often, the derived requirements are documented less formal, for example as notes, videos, audio transcriptions or mindmaps (Balzert, 2009, p.509).

We believe that our approach provides multiple benefits in this context, since it enables the described less formal representation formats, but gives the interconnections a structure and makes it explicit. The code system can be seen as the backbone that interrelates all artifacts, yet does not limit the representation formats.

3.1.5.3 The Software Requirements Document (SRS)

The SRS is a document which contains a complete collection of all requirements of the project. It arises as a result of the RA-process in general and the requirement specification phase specifically, which is described in section *3.1.4.4 Requirements Specification*.

The SRS states what the system developers should implement. This will include both the user requirements for the system and the system requirements, which will define the specifics in detail (see also section *3.1.3.1 Abstraction Levels*).

Corresponding to the quality criteria for requirements, there are also 8 characteristics of a high-quality SRS, which are defined in (*IEEE29148.2011*, 2011, sec.5.2.8, pp.12ff.):

- **completeness** For each desired functionality of the system, all possible inputs, events and the demanded reactions of the system have to be documented. This includes also the non-functional requirements like i. e. availability. The SRS must not include incomplete requirements.
  The completeness of the SRS often constitutes the greatest challenge of requirements analysis (Rupp & SOPHISTen, 2014, p.28f.).
- **consistency** The SRS must not include conflicting requirements or particular requirements multiple times. Furthermore the technical terminology of the domain needs to be consistent; the same entity must be labeled with the same identifier throughout all requirements (Rupp & SOPHISTen, 2014, p.29).
- **correctness** relative to the collection of requirements
- **unambiguity** allows exactly one interpretation
- **evaluation of importance and/or stability**

- **verifiability** allows the verification of the implementation of the SRS in an efficient way
- **modifiability** particularly includes the lack of redundance
- **traceability** "Requirements traceability refers to the ability to describe and follow the life of a requirement, in both forwards and backwards direction (i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)" (Gotel & Finkelstein, 1994).

It is advisable to use a standard structure for writing a SRS (Ludewig & Lichter, 2013, p.393). There are various templates available which were developed and have proved their worth in years of engineering work. A prominent example can be found in (*IEEE29148.2011*, 2011, chapter 8).

3.2 openETCS

We used the openETCS project as an example project to execute this new approach. This section provides an overview of openETCS. A more detailed description can be found in (Hase, 30.03.2012) and the full project proposal can be found at (Hase, 04.07.2013).

openETCS is appendant to the European railway sector and is the successor of the ETCS project. The goal of ETCS is to replace the national and proprietary signaling systems in 30 countries. The project aims at the development of on-board units that make trains which are equipped with ETCS capable to adopt to each signaling system in Europe. Furthermore ETCS on-board units shall be the standard equipment in future trains.

By harmonizing and unifying the different systems various benefits are expected: the reduction of national boundaries, increase of competition and in general decreasing costs for the European railway systems of the future.

Yet, more than two decades after the start of ETCS, not a single on-board unit is authorized for all European tracks, although more than 4000 km of track are equipped with ETCS and a detailed technical specification is available to the public (Hase, 30.03.2012). The project is afflicted with several difficulties: The high complexity of a multi-national project in combination with lots of national specialties, a imprecise specification and the manageability of highly complex, proprietary software products, which also entail in high follow-up costs.

The openETCS project is designed to tackle these difficulties.

The key aspect of openETCS is the use of the open Proofs concept. This means that not only the end product software, but every document and tool that is used for specification, development, operation and maintenance need to be FLOSS licensed, which aims at building an open source consortium and at having a strong standardizing effect. The channeling of experts is expected to result in higher quality and reliability, with decreasing costs at the same time.

The goal of our example project was to find out about the tool chain of openETCS which is used at the requirement analysis phase of the project. We did a requirement analysis of the RA phase of openETCS and used a qualitative data analysis approach which is based on Grounded Theory, but substantially modified. This approach is described in detail in chapter *2.4 Research Approach*.

3.3 Grounded Theory

In this thesis we propose an adapted transformation of Grounded Theory. The following chapter introduces the original Grounded Theory approach. The adaption is discussed in chapter *2.4 Research Approach*.

3.3.1 Qualitative Data Analysis

Qualitative research is a discipline from social studies which preoccupies with the investigation and analysis of so-called qualitative data. It is focused on the description, interpretation and understanding of behavior and interdependencies.

Qualitative data are non-numerically measurable data which often present themselves in continuous text or in audiovisual forms. As they are not processable with common standardized techniques (like i.e. mathematical methods), the research methods are mostly flexible and open.

Qualitative research is employed at fields of study where no structures or theories are present that allow a deduction of hypotheses. It aims at finding general rules and therefore derive explanations and an understanding for the field of research.

3.3.2 Grounded Theory

Grounded Theory is a "qualitative research method that uses a systematic set of procedures to develop an inductively derived grounded theory about a phenomenon" (Strauss & Corbin, 1990, p.24), which is originally developed by (Glaser & Strauss, 1967). It is one of the standard methods of methodically controlled qualitative research.

Today, there are various schools of thought concerning Grounded Theory, which developed out of the diverging understanding of the originators themselves. These different approaches will not be discussed in this thesis since they are beyond its scope. This thesis will follow the methodology proposed by (Strauss & Corbin, 1990).

Grounded Theory is an iterative process with continual elicitation of new data which are coded and analyzed in parallel. This means, in every iteration data is collected, coded and processed and this data serves also as an input for deciding how to progress further on.

This particularly distinguishes Grounded Theory from other techniques, where in a first step huge amount of data is collected and in a second step this data is analyzed.

3.3.2.1 Coding

Coding is the key process in Grounded Theory (Bryant & Charmaz, 2007, p.265). The task of data coding is to split the data up into units of meaning, ordering them into categories and sub-categories and finally derive a theory about the research topic by building a well-structured, hierarchical arrangement of categories.

The coding process consists of three coding steps: Open Coding, Axial Coding and Selective Coding.

3.3.2.1.1 Open Coding Open Coding is the initial coding step during the data analysis in Grounded Theory and is "the process of breaking down, examining, comparing, conceptualizing and categorizing data" (Strauss & Corbin, 1990, p.61). The new data, like i.e. an interview transcript, is analyzed and split up into units of meaning, thus statements, ideas, or phrases. Each unit is labeled with a so-called *code*, which represents the content of the excerpt or an annotation.

There are four components which build a theory when using the Grounded Theory approach:

- **codes** represent the content of the associated phrase. They should be selected logically and represent the data.
- **concepts** are collections of codes with a similar content. This allows the data to be grouped.
- **categories** denote phenomena. They are collections of similar codes. They serve as a first level of grouping the codes and deriving a theory.

- **properties** are attributes of a category.

3.3.2.1.2 Axial Coding

The term axial coding labels the process of ordering the data which were fractionated during the open coding step. By comparing the similarities and differences between the concepts from the open coding step, they are ordered into categories, which are more abstract.

To help the researcher with a systematic thinking about the data, (Strauss & Corbin, 1990) provide the so-called *coding paradigm*, which focuses the relationships between the concepts to the following aspects: causal conditions, phenomenon, context, intervening conditions, action/interaction and consequences (Halaweh, 2011, p.55).

3.3.2.1.3 Selective Coding

Selective Coding is the final of the three coding steps and applies the definition of core categories. The categories are rarefied at a high level of abstraction and those categories that appear repeatedly are candidates for core categories. In the end, the step of selective coding will lead to the setting up of a theory.

Therefore selective coding resembles axial coding, yet it is done on a more abstract level of analysis (Strauss & Corbin, 1990, p.117).

3.3.2.2 Memos

During the whole process of coding the researcher composes so-called memos. A memo is directly associated with a code. Memos are short texts that the author uses for recording ideas, arising questions etc. Therefore they support the coding process and ultimately the deriving of the desired theory. An example for this type of memo can be found in figure 2.3.

3.3.2.3 Theoretical Sampling

"Theoretical Sampling denotes the process of collecting data with the goal of deriving a theory. During the process the researcher collects, codes and analyzes data in parallel and decides, which pieces of information to elicit next and where to find them. This process is controlled by the theory to be developed" (Glaser & Strauss, 1998, p.53).

Hence, this technique aims at choosing new data sources in a way that enables a broader and deeper understanding. The desired result is that only important data is processed. Another result is that the volume of data to be analyzed is reduced.

3.3.2.4 Constant Comparison

The fundamental principle of data analysis in Grounded Theory is the constant comparative method. This means that after the researcher coded a particular phenomenon, he will systematically search for more data which either contrast or confirm the coded phenomenon.

Thereby, data become an indicator for a underlying concept which is denominated through the coding. During the further process, codings will then be mapped into concepts, out of which categories and ultimately core categories will be extracted (Mey & Mruck, 2007, p.25).

3.3.2.5 Theoretical Sensitivity

This term describes the ability of the analyst to recognize what is important in the hitherto data. Therefore it characterizes a personal quality of the researcher.

"Theoretical sensitivity refers to the attribute of having insight, the ability to give meaning to data, the capacity to understand and capability to separate the pertinent from that which isn't. All this is done in conceptual rather than concrete terms. It is theoretical sensitivity that allows one to develop a theory that is grounded, conceptually dense and well-integrated - and to do this more quickly than if this sensitivity were lacking." (Harvard Business School, 2007, p.41f.)

3.3.2.6 All is Data

One of the fundamental properties of Grounded Theory is that not only interviews or documents, but anything that is data can be used by the researcher at studying a certain domain or phenomenon. Everything that helps the researcher generate concepts for the emerging theory is a valid data source, including i.e. meeting protocols, newspaper articles, films, television talk shows, law texts, group meetings etc.

3.3.2.7 Theoretical Saturation

Theoretical saturation means that "no additional data can be found which helps the researcher to develop further properties of the category (Glaser & Strauss, 1998, p.69). When all categories are saturated, the elicitation of data is finished.

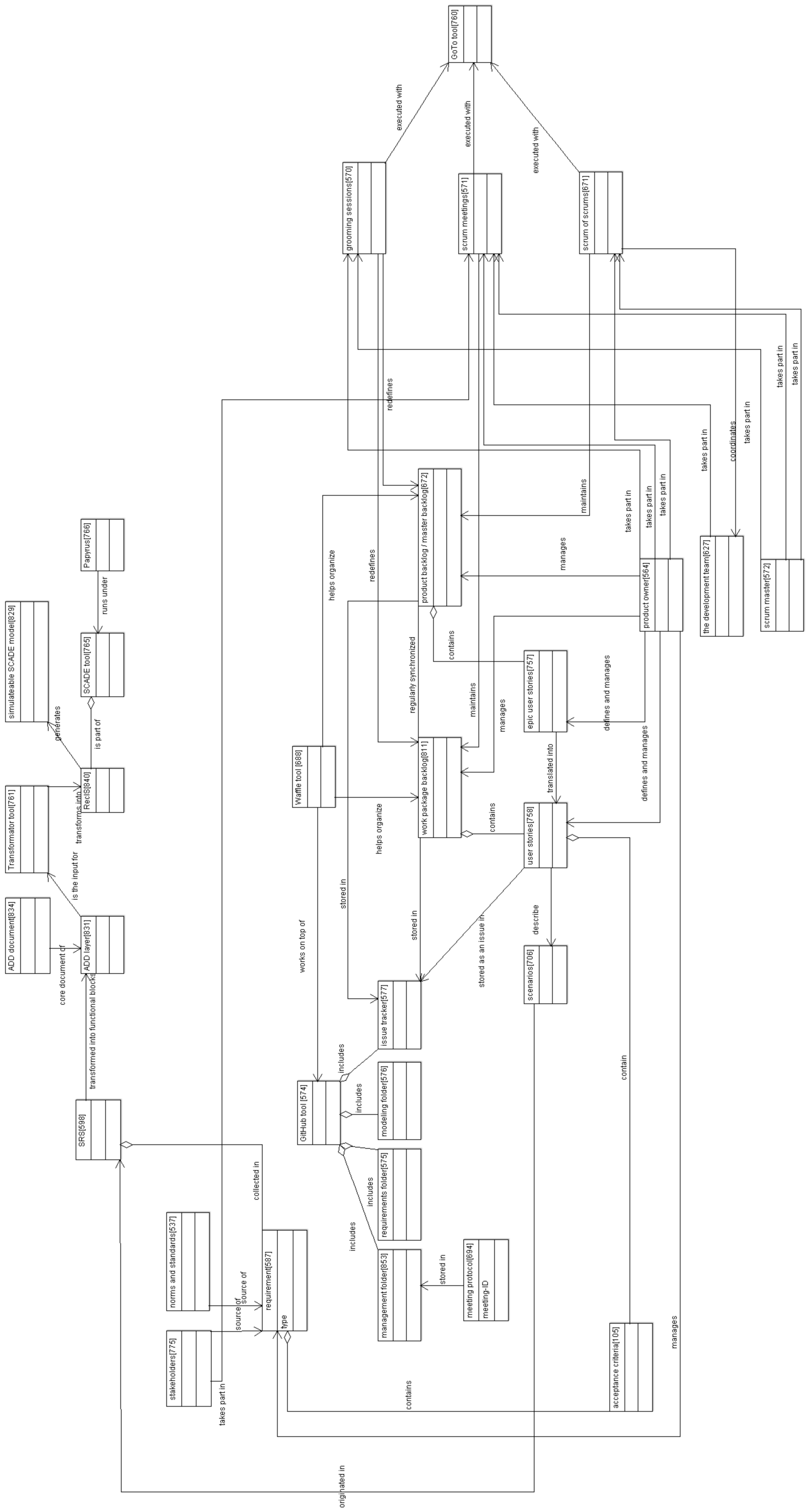

Figure 3.7: The derived Domain Model of the RE tool chain of openETCS

Bibliography

Ambler, S. W. (2008). Beyond functional requirements on agile projects. *Dr.Dobb's Journal*, *33*(10), 64–66.

Balzert, H. (2009). *Lehrbuch der softwaretechnik: Basiskonzepte und requirements engineering* (3. Aufl. ed., Vol. 1). Heidelberg: Spektrum Akademischer Verlag.

Booch, G. (2006). *Object oriented analysis & design with application*. Pearson Education India.

Braude, E. J. & Bernstein, M. E. (2011). *Software engineering: Modern approaches* (2nd ed. ed.). Hoboken, NJ: J. Wiley & Sons.

Bryant, A. & Charmaz, K. (2007). *The sage handbook of grounded theory*. Los Angeles and London: SAGE.

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.

DIN. (2009-01). *Projektmanagement - projektmanagementsysteme - teil 1: Grundlagen* (69901-1: ed.) (No. 69901). Berlin: Beuth.

Dumke, R. (2001). *Software-engineering: Eine einführung für informatiker und ingenieure: Systeme, erfahrungen, methoden, tools* (3., überarb. Aufl. ed.). Braunschweig and Wiesbaden: Vieweg.

ESI. (n.d.). Eight things your business analysts need to know: A practical approach to recognising and improving competencies.. Retrieved 08.05.2015, from `http://www.esi-intl.com/resources/knowledge-center/thought-leadership/business-analysis/eight-things-your-business-analysts-need-to-know`

Glaser, B. G. & Strauss, A. L. (1967). *The discovery of grounded theory: Strategies for qualitative research*. Chicago: Aldine Pub. Co.

Glaser, B. G. & Strauss, A. L. (1998). Grounded theory. *Strategien qualitativer Forschung. Bern*, 53–84.

Gotel, O. C. Z. & Finkelstein, A. C. W. (1994). An analysis of the requirements traceability problem. In *Requirements engineering, 1994., proceedings of the first international conference on* (pp. 94–101).

Halaweh, M. (2011). Using grounded theory as a supportive technique for system requirements analysis. In *Icons 2011, the sixth international conference on systems* (pp. 54–59).

Halaweh, M. (2012). A case study of using grounded theory-based technique for system requirements analysis. *Journal of Information Systems and Technology Management*, *9*(1), 23–38. doi: \url{10.4301/S1807-17752012000100002}

Harvard Business School. (2007). *Basics of qualitative research.* Retrieved from `http://isites.harvard.edu/fs/docs/icb.topic536759.files/Strauss_Corbin_Chaps_3_and_4.pdf`

Hase, K.-R. (04.07.2013). *openetcs - full project proposal: Open proofs methodology for the european train control system.* München.

Hase, K.-R. (30.03.2012). *openetcs projektsteckbrief: openetcs soll europäische zugsicherung interoperabel, sicher und bezahlbar machen.*

Hickey, A. M. & Davis, A. M. (2003). Elicitation technique selection: how do experts do it? In *Requirements engineering conference, 2003. proceedings. 11th ieee international* (pp. 169–178).

IEEE. (n.d.). *Ieee 610.12-1990* (610.12-1990 ed.). IEEE Press. doi: \url{10.1109/IEEESTD.1990.101064}

Ieee29148.2011 (First edition, 2011-12-01 ed.). (2011). Geneva and New York: ISO and IEC and Institute of Electrical and Electronics Engineers.

Kaufmann, A. & Riehle, D. (2015). Improving traceability of requirements through qualitative data analysis. In *Software engineering.*

Ludewig, J. & Lichter, H. (2013). *Software engineering: Grundlagen, menschen, prozesse, techniken* (3., korrigierte Aufl. ed.). Heidelberg: Dpunkt.verlag.

Marciniak. (1994). Encyclopedia of software engineering.

Mey, G. & Mruck, K. (2007). Grounded theory methodologie—bemerkungen zu einem prominenten forschungsstil. *Historical Social Research/Historische Sozialforschung. Supplement*, 11–39.

Prime Minister's Strategy Unit. (2004). Strategy survival guide. *Cabinet Office, Admiralty Arch, The Mall, London SW1A 2WH*.

Rupp, C. & SOPHISTen, d. (2014). *Requirements-engineering und -management: Aus der praxis von klassisch bis agil* (6., aktualisierte und erweiterte Auflage ed.). München: Hanser, Carl.

Sommerville, I. (2011). *Software engineering* (9th ed ed.). Boston: Pearson.

Sommerville, I. (2012). *Software engineering* (9., aktualisierte Aufl ed.). München: Pearson.

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

van Lamsweerde, A. (2009). *Requirements engineering: From system goals to uml models to software specifications.* Chichester, England and Hoboken, NJ: John Wiley.