Technical Reports Mathematics/Computer Science
FB IV - Department of Computer Science
University of Trier
54296 Trier, Germany

Grounded Requirements Engineering:
An Approach to Use Case Driven Requirements Engineering

David Würfel, Rainer Lutz, and Stephan Diehl

Technical Report No. 14-2 July 2014

Abstract

Context: Requirements engineering is a difficult but very essential step in software development. It produces a specification of the needs or conditions to meet for a software product. This specification may be vague and ungrounded, i.e. the relation of the requirements to the observations they are derived from may be unclear or not documented. Furthermore, stakeholders may be influenced by solutions of existing software without knowing if these actually suit the software to be developed.

Objective: In order to cope with the above issues, it is important to understand the complete task or problem, before designing a software system to support or solve the task. Thus, we developed a method called Grounded Requirements Engineering (GRE) that leverages the Grounded Theory methodology to observe and analyze processes and user activities in the real world.

Method: GRE is an iterative process consisting of two steps. In the first step, Grounded Theory methods are used to analyze user experiments or interviews. In the second step, the resulting abstract descriptions of the user behavior are transferred into use cases, a widely used requirements representation. GRE produces complete, comprehensible, and grounded requirements for the software system to be built, i.e. the requirements are traceable back to their origins.

Results: In this paper, we provide an elaborate description of the GRE method and illustrate it by applying it to derive requirements for an interactive software tool for model merging.

Conclusion: GRE is a systematic approach for eliciting, documenting, and maintaining requirements. It combines established methods from social sciences and software engineering. We argue that it fills a gap in the arsenal of elicitation methods of today's requirement engineers.

Grounded Requirements Engineering: An Approach to Use Case Driven Requirements Engineering

David Würfel, Rainer Lutz, and Stephan Diehl
{s4dawuer, lutzr, diehl}@uni-trier.de

July 17, 2014

1 Introduction

Requirements engineering (RE) is a difficult and very essential step in software development. Regardless of which process model one follows—an agile or rather strict and traditional one—requirements engineering is the most critical phase in software development [Boe81, NK91]. Traditionally, requirements are derived in a discursive approach with customers and other stakeholders. Through brain storming sessions, interviews, or questionnaires one tries to elicit information about the customers' needs and based thereupon the requirements for the future software system. There are three issues that often occur in requirements elicitation:

Unawareness (I1): Requirements elicited from customers are often vague and do not provide detailed information on the software system to be developed. In particular, this happens for novel tasks that have not been supported by software before [NE00].

Early Commitment (I2): Customers or designers may be influenced by existing software, just follow the beaten paths, and simply reuse previous solutions or patterns without knowing if these perfectly suit the software to be built [Hal11].

Groundedness (I3): All stakeholders should be able to reconstruct an understanding of how a requirement was produced [GF94]. To this end, the requirements should be grounded in the reality and linked to their underlying observations. This is a prerequisite for requirements traceability—"the ability to describe and follow the life of a requirement in both forwards and backwards direction" [GF94] through the whole software development process.

The term *requirements* is often defined as capabilities or externally observable characteristics of a desired system [IEE90, Dav05]. In order to cope with the problems above, it is important to understand the complete task or problem, before actually drawing a line between the user and the desired system [NE00].

In recent years, researchers have realized that there is a connection between RE and *Grounded Theory* (GT) [GS67, SC08], a well known and widely applied qualitative methodology from the social sciences. Berry et al. [BGH+08] note that the GT "provides a systematic description of the processes of requirements engineering and architecture recovery, which might otherwise seem to be random searches." In the cognitive and social sciences there exists a large number of papers and books providing advice on how to systematically analyze data using the GT. While some software engineering researchers explicitly mention in their papers that applying the GT to derive requirements is straightforward (see Section 6), we observed this transfer to be less obvious.

In this paper we propose *Grounded Requirements Engineering (GRE)*, a novel, systematic approach for RE that addresses the three issues discussed above (I1,I2,I3). GRE makes extensive use of the Grounded Theory. In a nutshell, GRE is an iterative process. First, it uses the GT for the actual sampling and analysis of recorded user experiments or interviews. The result of the analysis is an abstract description of the user behavior. In a second step, the results are transferred into use cases, which distinguish system and user behavior and provide complete, comprehensible, and grounded requirements for the software system.

To the best of our knowledge, we are the first to describe a whole GT-based requirements engineering process, which turns the results of the GT method into an established requirements representation in form of uses cases.

2 Background

Grounded Requirements Engineering is based on two established methods from different disciplines; the Grounded Theory from the social sciences and the Use Case Model from software engineering. As Cheng et al. already suggested, to overcome current issues in RE, researchers have to "seek out collaborators from other disciplines to leverage successful techniques" [CA07]. At a first glance, the GRE approach might appear unorthodox but even "at a risk of possible failure researchers need to think beyond current RE and SE knowledge and capabilities" [CA07]. Therefore, we believe that GRE is a novel and inspiring contribution to requirements engineering.

2.1 Grounded Theory

Grounded Theory [GS67] is a qualitative research methodology introduced by Barney Glaser and Anselm L. Strauss in 1967. Coming from the social sciences its purpose is to create a theory tightly connected to the data obtained by observing human factors including social behavior and interaction. Note that both the emerging inductive theory as well as the method to build the theory itself are called Grounded Theory. Over the years, two variants of the original methodology evolved. Here, we focus on the GT method [SC08] extended by Anselm Strauss and Juliet Corbin who suggested a less strict approach that can be better applied in practice. Typically, the GT method is used to analyze interview transcripts, but it also works for transcribed video recordings.

Although the GT method was originally developed and applied in the social sciences, it has also been used in software engineering [All03, HNM10] or human computer interaction [FBC11] to examine human factors. For instance, Begel and Simon [BS08] applied the GT for a long-term analysis of behavioral patterns of professional novices, Crabtree et al. [CSN09] used it to investigate how humans describe software processes in natural language, and Dabbish et al. [DSTH12] analyzed the social interaction in GitHub, an open software-hosting repository.

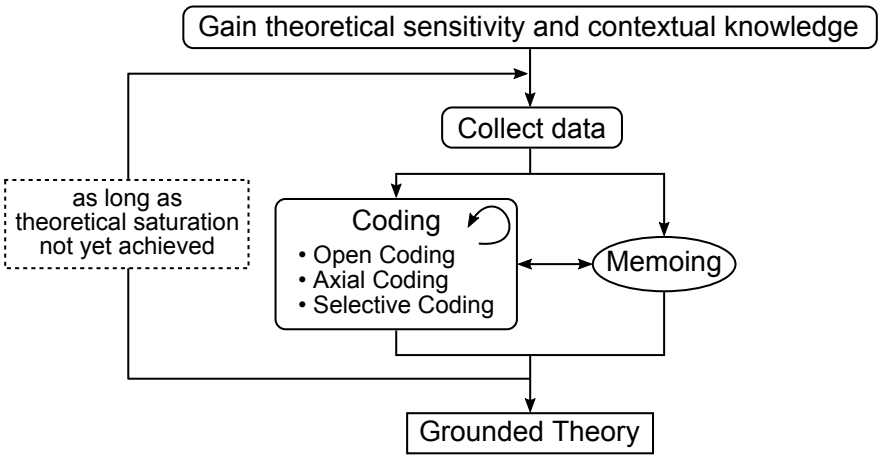

Figure 1: Typical procedure of the GT methodology

The GT method can be varied and is not intended to progress in a strict way. However, we depict a typical procedure in Figure 1. In order to apply the GT, the researchers must at least have the required contextual knowledge and the so called *theoretical sensitivity*. This means the ability to understand the

elicited data and assign a meaning to it; so to conceptualize and formulate a grounded theory. The actual research question can and should be rather open because concrete insights manifest as a result via applying the GT.

The collected data, often gathered through interviews or observations of the real world, is analyzed by an activity called *coding*; an interpretative analysis of transcribed observations. The various coding types are not necessarily sequential and may be repeated or applied in parallel.

Open coding: Transcriptions are processed via a line-by-line analysis and annotated with so called *concepts*. These concepts name observed phenomena or activities. Thereafter, they are grouped and assigned to abstract terms, so called *categories*, which comprise multiple concepts.

Axial coding: In this step the previously defined categories are further elaborated by defining properties of each category. Additionally, codes (categories and properties) are related to each other, via a combination of inductive and deductive thinking. Thus, a taxonomy of categories and subcategories arises.

Selective coding: This coding step is typically done at last. A *core category* is identified and all the other categories are aligned to it. The researchers are now able to finally develop a conceptional dense theory—the Grounded Theory. The core category may have the most relationships and serves as a basis or central idea of the overall theory. It should provide the important parts of the investigated problem.

In parallel to those codings, *memos* are produced to note the researchers' thoughts. Memoing supports the structural process of the GT because the researchers are repeatedly reflecting on their emerging ideas. The drawing of diagrams can also be seen as memoing. During the whole GT a *constant comparison* is applied, i.e., emerged observations, concepts, categories and their relationships must always be compared to and validated against previously known ones. This idea is an integral component of the GT and supports the grounding of the theory in the data. Also, *theoretical sampling* may be used to start analyzing the data even when the data collection is not completed. It describes the activities of gathering, coding, and interpreting the data to guide the selection of further data. Data gathering, coding, and memoing are cycled until the *theoretical saturation* is achieved—a state in which new data does not lead to more insights and developing of the grounded theory is completed for the time being. The GT has a high conceptual density when it is coherent, comprehensible, and grounded, i.e., it can be validated in the data.

The result of the GT is a complex network of concepts, categories, subcategories, their properties, and an *analytic story*. This story must be created by the researchers from the taxonomy of the categories, their memos, and all insights. It is intended to describe the relationships and the emerged theory as precise, concise, and intersubjective as possible, i.e., one can understand the overall result of the GT study without detailed information.

2.2 Use Cases

Use cases describe the behavior of a software system under different conditions and interactions with its user(s) in specific roles. They are not only intended for documentation purposes, but may also serve as a basis for communication with stakeholders and for further requirements analysis.

On the one hand, the Use Case Model comprises the use case diagram as a part of the UML notation, which shows structural relationships between use cases and involved actors. It is primarily designed to give an overview of the systems behavior and scope. At a certain complexity, such a diagram must not be used for communication with the costumer without further explanation. Therefore, the Use Case Model, on the other hand, includes use case descriptions or often simply called use cases. These are a textual description of an interaction between the user and the system in rather simple natural language. In general, it contains a name, a primary actor, and a short description including its goal. Depending on the level of detail there may also be triggers, results after successful interaction, pre- and postconditions, and a detailed main success scenario. A scenario is a specific interaction sequence whereas the main success scenario specifies the most frequently and likely case. Ideally, the use case would contain all possible alternative scenarios, e.g., on occurrence of errors or special cases.

For a use case diagram the UML specification delineates the notation of the diagrams whereas for the use case description there currently is no defined standard. However, the frequently cited work from Alistair

Cockburn [Coc00] may serve as a *de-facto* standard for use case descriptions. Cockburn recommends to decide on a rather casual or strict form of uses cases depending on the project context. For GRE we suggest to extend the strict form, consisting of fifteen fields, with two additional ones for special purposes as shown for the example in Table 4.

3 Grounded Requirements Engineering

In a typical context of Grounded Requirements Engineering it is not obvious what requirements for a future system look like (I1). One simply does not want to rely on the customer like in a classical RE approach where requirements are elicited through questionnaires, interviews, or brainstorming sessions or even on existing software or solutions (I2). Instead, the real physical world is investigated in an unbiased manner and gained insights should later be supported by the software. Furthermore, it is crucial to preserve the groundedness (I3), i.e., one should always be able to trace a result back to the underlying observations.

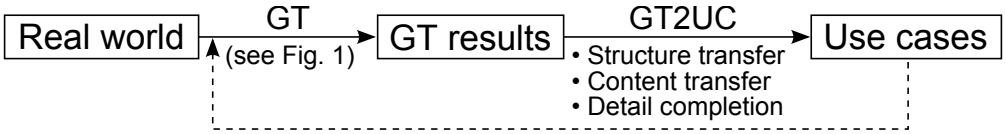

Figure 2: Grounded Requirements Engineering

The general process of the GRE approach is depicted in Figure 2. GRE is intended for rather complex and novel problems for which at least most of the requirements are not obvious but can be observed in the real world. Here, real world observations do not necessarily describe activities without any technical support as in our case study in Section 4.2. It can also involve interactions with non-specialized software or even hardware devices. However, it is important to capture the reality as completely as possible, for instance, via video recordings or screen captures. Next, the GT method is applied to conduct a systematic and structured data analysis and derive a theory or fully detailed description of observations. The result of this analysis is a complex network of interrelated categories and subcategories—the so called Grounded Theory—a profound description of processes in the real world. Typically, not every stakeholder has knowledge about the GT methodology. Therefore, the GT results must be transferred to a standardized format—the Use Case Model—in order to represent requirements in a form which is used in many existing software development processes.

3.1 Applying the GT method

The GT method as described in Section 2.1 is very general. For our approach we use a well defined and specific instance of the GT method. In GRE data collection means observing users solving tasks. Thus, we can gain deep insights into how a given problem is tackled. The categories emerging from the coding phase concretely map to both physical and mental activities that are performed by the users. Mental activities describe those that cannot be observed visually but may be captured by thinking aloud methods [ES80, Nie93]. This allows us to collect information about otherwise overlooked requirements. Theoretical sampling is performed by choosing another set of users and by defining new tasks or refining existing ones. The rest of the GT method can be conducted like proposed by Strauss and Corbin [SC08].

3.2 Transfer from GT results to Use Cases

In the next step, the complex network of categories resulting from the GT must be transferred to the standardized Use Case Model. In this paper we refer to this process as GT2UC and to the requirements engineers in charge as the GT2UC team. Since the GT results capture activities of the participants, we think that use cases are best to hold this kind of information. Inspired by the use case evolution of Kulak

and Guiney [KG00] the sub-process of the GRE itself is divided into multiple passes: structure transfer, content transfer, and detail completion. In every pass the use cases are filled with more and more details. After the first two passes the use cases are still described in terms of a physical perspective (participants and physical artifacts), whereas the detail completion pass switches to the software perspective (users and software system). Each pass has different properties as shown in Table 1.

Table 1: Properties of the GT2UC passes. Background colors represent different types of information (see Section 4).

	Structure transfer	**Content transfer**	**Detail completion**
Source	explicitly from GT results	implicitly from GT results	external or implicitly from GT results
Transfer	direct	interpretative	discursive
Type of information	behavior and conditions	behavior, conditions, progress information, interaction properties, modifications of physical artifacts, relevant details	information missing from GT results (emerging from the software perspective)
Perspective	physical	physical	software

3.2.1 Structure transfer

At first, during the structure transfer only initial use cases are created. This step should bring up a rough structure for the requirements and is intended to identify nearly all involved use cases. The GT data is transferred as explicitly as possible and with as less interpretation as possible.

Each category from the GT results may contain different types of information (cf. Table 1), which have to be identified by the GT2UC team and contribute to the requirements in different ways. In this pass, information concerning behavior and conditions are most important whereas all other types of information are rather not taken into account. Behavior means, for instance, observed behavior of participants or possible behavior of a future software system. Conditions in category descriptions may be easily identified by keywords (*already*, *based on*, *before*), specific use of perfect or past tense, or references to already existing use cases.

In order to derive the use cases, the GT2UC team should begin with the core category and systematically work through the network of categories in a depth-first manner, i.e., each subcategory is examined after its parent category. Therefore, they simply decide whether a new use case emerges from a subcategory or whether it has already been covered by the use case derived from its parent category. In contrast, use cases derived from parent categories might also be declined because specialized use cases from subcategories may have no need for an abstract, parental use case. For initial use cases only the following fields are relevant:

Use Case ID and Name: The name should map the title or functional content of the category. Thus, one can directly infer the relation to the category the use case emerged from. As described by Cockburn [Coc00] this name should be an active verb-goal phrase that names the goal of the primary actor.
Description: Contains explicitly transferred information about behavior from the GT result.
Primary actor: Is directly transferred from the GT results, e.g., the participant.
Pre-/postconditions: Contains information about conditions as identified in the GT results.
Associated information: All categories that are involved in this use case are noted in this field in order to maintain traceability.
Change history: For logging and versioning purposes, the date, the current GT2UC pass, and optionally, the reason why a change was made is listed.
Open issues: In the first pass the GT2UC team should not worry too much about open issues but some may be recorded even in this early phase.

Once all categories along with their subcategories are covered, the GT2UC team has to check whether it makes sense to combine some of the newly formed use cases. This helps to keep the number of use cases well-arranged (as there may emerge a use case from each of the many categories). In particular, use cases from parent categories often do not contain significant detail or may be too abstract. In contrast, use cases evolved from subcategories may only differ slightly and thus combining them seems more reasonable.

Additionally, a ***glossary*** is maintained. It contains specific terms that result from the GT and need to be defined or described in more detail.

3.2.2 Content transfer

In this pass the initial use cases, which resulted from the structure transfer, are reconsidered and enhanced. This should be done in such a way that all information from the GT results will be contained in the use cases thereafter. To this end, the GT2UC team has to interpret the GT results because many terms and definitions must be aligned to the use case format. At the end, the interaction described by a use case should be short, precise, and represent active behavior from the primary actor's point of view. Note that some information is not directly present in the category description. The GT2UC team has to carry out these interpretations carefully to prevent false assumptions and to preserve the groundedness of the theory.

During the second pass not only behavior and conditions are considered, but also all information from those categories listed in the field *Associated information* (cf. Table 4). This can be classified as follows: Progress information characterizes relations between use cases or the sequential ordering of interactions. Interaction properties refer to complexity, importance, or frequency of interactions. Modifications of physical artifacts give hints to design considerations for user interfaces. Finally, there may be relevant details that provide additional information. In contrast, some information might be irrelevant (e.g. examples of a participant's behavior) or redundant, and therefore, can be omitted in the use cases. In order to transfer all relevant aspects of a category, we extended the use case format of Cockburn [Coc00] by two additional fields to record *Priorities* and *Modifications of physical artifacts*. Thus, the information gathered from the category descriptions is transferred to the fields of the use cases as follows:

Description: The description of the use case may be enhanced with progress information or adjusted to be more precise.

Scope: Typically, the scope of the interaction is not explicitly mentioned in the GT results and thus has to be interpreted by the GT2UC team.

Level: Cockburn distinguishes three levels: *summary*, *user*, or *subfunction*. The user level describes an interaction of the primary actor whereas the summary level is a rather abstract description of an interaction. Subfunction use cases are very low-level and rather describe *how* an interaction is done than *what* it consists of.

Stakeholders and interests: This field specifies all actors affected by the interaction of the underlying use case. Often, the GT2UC team has to interpret the information from the respective categories.

Pre-/postconditions: More implicit conditions that are not directly contained in the GT results may be added, e.g, conditions that naturally result from the context of the GT data acquisition, i.e., the study itself.

Success guarantees: The results after successful execution of the main success scenario are noted in this field. In general, these artifacts are goals of the actors or preconditions for further interactions.

Triggers: This field mostly correlates with the interest of the actors or characterizes states, which introduce the actual interaction after all preconditions are fulfilled.

Main success scenario: This is the most important field of the use case and contains a stepwise description of each action from the overall interaction sequence. Note that these steps can be executed sequentially or in parallel. Mostly, the scenario has to be interpreted based on the category description. Each step is enumerated and should describe a single and complete action from the actor's point of view. The main success scenario characterizes the most common case and elaborates on the interaction outlined by the use case description.

Extensions/alternatives: This use case field covers exceptional cases of the main success scenario, e.g., defined by concrete properties of a category. Usually, it has to be interpreted by the GT2UC team.

Priority: After choosing an appropriate priority scale, interaction properties in the category description often define priorities of use cases, e.g., use cases emerged from the core category should typically have a high priority. Sometimes, vague keywords in the category description like *essential* or *very important* express priorities, which may be at least treated interpretatively.

Modifications of physical artifacts: This information is directly transferred from the GT results to the according field. Later, it may be used to justify specific design decisions with respect to the user interface, and therefore, it must be captured independently from the actual interaction (the main success scenario).

Miscellaneous: Information that is somehow relevant but cannot be recorded elsewhere.

Additionally, use cases may have some unstructured and strange syntax which results from the direct transfer of the first pass and must be adjusted here. For instance, the description of the use cases can be cleaned by removing redundancies and changing the syntax to the proposed use case format, i.e., short, active descriptions of interactions from the actors point of view. Moreover, the main success scenario, which will be derived during this pass, may further be used to gain a more precise use case description.

After each use case has been edited, cross references to other use cases are identified and marked, for instance, by underlining. Thus, the GT2UC team quickly gains an overview about the detailed structure of the use cases and opportunities for combining use cases may become more obvious. Additionally, at this stage a use case diagram may help to visualize these cross references by inclusion, extension, or even generalization relations and thus provides a very good overview of the observed participant behavior. At the end of the second pass all relevant information from the GT has been transferred to the use cases.

3.2.3 Detail completion

The detail completion pass includes switching from the physical to the software perspective. At this stage, the system boundary is defined, i.e., the software system along with its functionalities and users is introduced. Participants and their activities should be replaced by users, user activities, and software functionalities. Moreover, there may be some missing use cases, which are needed for a software system but were not part of the GT study.

Although at this point all information of the GT results is contained in the use cases, there may be open issues that require another information gathering iteration. This can either be achieved by reexamining the GT results or by more traditional forms of RE. In the former case, missing information is explicitly sought in the GT results to reveal otherwise overlooked features. Traditional methods from RE can be useful if it seems too time consuming to go back to the GT results or if the missing information can probably not be obtained from the GT in general. Also, the GT2UC team may not have sufficient expertise to solve some of the open issues on its own. Decisions about feasibility or external constraints like legal requirements are typically made by other domain experts. Since use cases are a widely used representation of software requirements and may also facilitate the communication with stakeholders, they can serve as a basis for further discussions, interviews, or questionnaires.

In order to achieve the three main goals listed below, during the detail completion pass all use cases are traversed again. This is best done in a bottom up manner, i.e., from user-level to summary-level use cases.

Identify missing use cases: A complete use case model should cover all user-system interactions of the software to be developed. In particular, this also comprises requirements that do not result from the GT study itself but are needed for a typical software system.

Resolve open issues: Open issues can either be resolved by revisiting the GT results or by more traditional RE approaches in conjunction with the use cases derived during the previous passes.

Maximize automation: With respect to usability it has to be decided which actions are executed by users and what can be done automatically by the software system. Typically, these decisions cannot be made by the GT2UC team alone, but have to involve all stakeholders.

According to Cockburn [Coc00] the main scenario should not contain more than ten steps or otherwise should be broken into several use cases. Hence, use cases may be split or combined in this pass. Moreover, the use case diagram that could have already been created during the content transfer may be adjusted accordingly. Note that detail completion might be time consuming because the GT2UC team probably has to revisit the GT results again. However, at the end the use case model should contain fully dressed use cases with traceability links to the GT results (I3). These use cases define requirements tightly connected to the observations from the real world while preventing unawareness (I1) and early commitment (I2) to a great extent.

3.3 Refining the GT results

Even after completely stepping through all three passes of the GT2UC transfer in rare cases there might be some shortcomings. This can be due to problems during the conducted user experiments or an improper application of the GT. If sufficient resources are available, additional user experiments may be useful to solve open issues or remove incomplete arguments. As this can mean another time consuming iteration, it should really be an exception rather than the rule.

4 GRE in Practice

In order to assess the usefulness the GRE method, we applied it to derive requirements for an interactive software tool for model merging. Imagine a scenario where there is an existing software project with its static entities being represented by a UML class diagram. A new software artifact, also represented by a UML class diagram, is developed in a different branch or bought from external resources. In order to integrate the new artifact and thus keep the design of the software system consistent, the two UML class diagrams have to be merged. However, there may be overlapping entities with the same names or even same semantics but different names. This is a non trivial task, which we believe cannot be achieved fully automatically. Thus, a tool could support the user to interactively merge both UML models. Requirements analysis for such a tool faces in particular issues I1 and I2 discussed in Section 1: the people involved in the requirements analysis may make decisions based on their experience with text merging tools and are further unaware of the actual steps involved because no elaborate interactive tool for merging UML models is available. This makes model merging an ideal candidate for applying GRE. Next, we briefly introduce the GT study and all necessary categories for the upcoming example of the GT2UC step.

4.1 Grounded Theory Study

In order to find out how humans merge UML models, we conducted a study where the participants were asked to compare and merge UML class diagrams [LWD11]. There, they only used pen and paper, which makes this process completely independent from any available software tools. For applying the GT method we videotaped all experiments and transcribed those videos. During open coding we identified over 180 concepts, which were assigned to categories or subcategories. Those were related to each other during axial coding. Finally, in the selective coding step the core category emerged and all the other categories were arranged around it—resulting in a complex network of categories and their subcategories (a simplified version is shown in Figure 3). The six top level categories contain about 75 subcategories in total. From this network we deduced the so called analytic story (Table 2). This narrative gives a brief overview of the derived theory and describes the typical behavior of a participant when merging UML class diagrams.

In order to illustrate the GT2UC step of our approach, we chose a single category from the GT results.

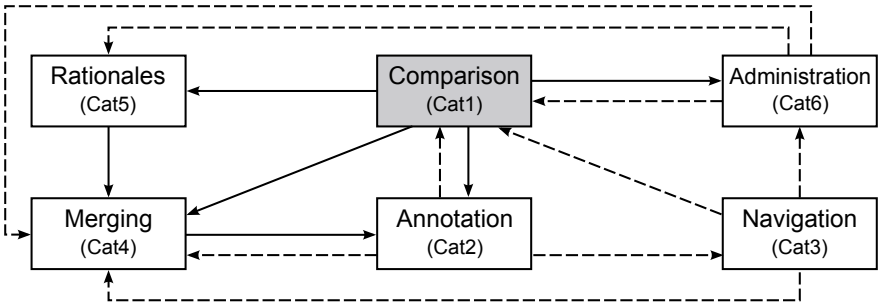

Figure 3: Relations between top level categories. Solid arrows encode preconditions and dashed arrows depict supportive relations.

Table 3 shows the textual description of the second category named *Annotation* along with one of its subcategories. The remaining subcategories are omitted because they are not part of this example. For more details on the GT results we refer to our previous work [LWD11].

4.2 Transfer from GT results to Use Cases

In the following we demonstrate how the second step of our approach—the transfer from the network of categories to the Use Case Model (GT2UC)—is accomplished. Due to space limitations we only explain this transfer for a single category (Cat2.4) in detail rather than providing an overview of the complete result or further tackling a combination and splitting of use cases explicitly. In particular, this example shows how a single use case evolves over the three passes. It represents a mostly common workflow, that is, the transfer of other use cases is similar. Table 4 depicts the sample use case after each of the three subsequent passes (middle column with (*) only, complete middle column, right column). There, different colors highlight the type of information that was transferred from the category descriptions (cf. Table 3).

4.2.1 Structure transfer

During the first pass only basic information of the category is transferred to the use case. Therefore, we created the use case itself, generated an unique identifier (UC-20), and adjusted the name of the category for the use case to match the naming conventions proposed by Cockburn [Coc00]. In order to obtain a rough outline of the use case, we searched the category for information that covers behavior or conditions, highlighted them as depicted in Table 3, and finally, copied them to the respective fields of the use case. Coming from the GT results the primary actor was defined to be the participant/subject. Furthermore, we noted all involved categories in the field *Associated information*. Since hints to the subcategories of Cat2.4 are already given in the use case description, we decided to cover them with the same use case. A list of all involved categories is not only useful for the subsequent passes, but also to maintain a certain traceability for the whole project. The result of the structure transfer for Cat2.4 is shown in Table 4. Here, only those fields marked with an asterisk (*) are part of this early use case. All the other fields are irrelevant for this pass.

Also, we added terms like "implicit and explicit realizations" or "distributed features" to a glossary along with a more detailed explanation, which could be derived from the respective subcategories. Once all the other categories were treated in a similar way, we searched for use cases that can be combined. In our example *UC-15 Point to an element* and *UC-16 Group elements* have been combined and renamed to *UC-14 Apply auxiliary annotations* (cf. Figure 4) because they only differed slightly and their behavior could be expressed in the parental use case.

4.2.2 Content transfer

In the second pass we transferred all relevant parts of the remaining information to the use cases. Therefore, we took all categories that are listed in the field *Associated information* and searched for different types of information. We also examined the top level category (Cat2). In particular, while browsing through all related categories, we highlighted those parts of their descriptions that cover progress information, interaction

Table 2: Analytic story of the model merging example.

Analytic story: After reading the requirements specification, participants started investigating both source diagrams in order to gather basic information about included features and the corresponding domain. Next, they compared the diagrams and concentrated on identifying similar elements, which was also annotated within the models themselves. Therefore, they navigated through both models based on the visually represented relationships. Similar elements were investigated in a more detailed step to discover further similarities and possible differences, and to study advantages and disadvantages. Based on this information the subjects developed rationales that helped combining the source diagrams. In addition, new/own ideas might be considered to improve or even extend the merged diagram. Finally, participants checked whether all elements have been investigated and some even validated the newly created UML class diagram [LWD11].

Table 3: Category descriptions with highlighted types of information (cf. Table 1).

<table>
<tr><td colspan="2">Cat2: Annotation. This category covers the annotation of matchings and differences and besides that more general strategies to record the collected information, which may support further comparisons or mergings. Participants tended to apply these techniques when the complexity of the diagrams increases whereas the level of detail of such annotations may vary and is depending on the processing habits of the participant.</td></tr>
<tr><td>Cat2.4: Annotation of Matchings: This subcategory comprises strategies to record observations on the similarity of two model elements (or groups of elements). While comparing both source diagrams, the participants used annotations to outline previously identified matchings. In particular, such matchings were highlighted based on a participant's individual strategy directly after they have been found. This may serve as a basis for later steps, e.g., during a more detailed investigation of the respective model elements or the actual merging activitiy. Subjects that did not annotate the source diagrams spent more time to recap previous observations, for instance, when they merged both models. Most participants used different colors, shapes, and/or line types to annotate matchings. Especially colors were used in two ways: different colors either indicated different matchings or different types of matchings. For the latter, participants basically distinguished between three types of matchings: (1) elements with the same UML type (e.g., class-to-class matchings), (2) elements that can be matched with more than one element (distributed features, 1:n matchings), and (3) explicit/implicit realizations, where the term explicit implies that additional classes were used to realize a certain feature, while implicit means that this can be achieved without them, e.g., by a simple object variable.</td><td>Cat2.4.1: Marking of two elements of the same type describes a linking of two similar elements of the source diagrams. Those are supposed to have the same type, i.e., both are classes, attributes, methods, relations or the like. Also, groups of elements may be marked as similar. Participants applied different strategies like the usage of colors or pointers, or the labeling of element groups with capital letters.

Cat2.4.2: Marking of distributed features generally considers the following case: one element in the first diagram is matched with multiple elements in the second one or vice versa. Often, an implicit grouping was applied, i.e., you can also speak of 1:n matchings.

Cat2.4.3: Marking of implicit and explicit realizations covers the situation when similar features in both diagrams were identified but have been designed implicitly in the first diagram and explicitly in the second one or vice versa. For instance, the usage of primitive datatypes in contrast to explicit classes. In general, this kind of annotation is based on 1:1 matchings, but was afterwards treated separately.</td></tr>
</table>

properties, modifications to physical artefacts, or other relevant details (cf. Table 3). Like in the first pass, we also searched for (additional) information about behavior or conditions.

On the one hand, some of the information could directly be transferred to the fields of the use case. For instance, the field *Modifications of physical artifacts* could easily be filled with information that describes in what way participants used colors, shapes, or links for annotation purposes. Note that this information may influence user interface decisions. However, a complete design of the user interface is part of subsequent development steps and thus is not a main goal of use case modeling. Also, the *Main success scenario* and a possible *Alternative* could be extracted from the categories explicitly. In our example these fields contain short scenarios as those are only dependent on the annotation strategies and the similarity types and do not have to describe a series of complex steps or alternatives. Nevertheless, especially summary-level use cases often provide more extensive scenarios.

On the other hand, there might exist fields that cannot be filled with explicit information. Simple examples are the fields *Scope* and *Success guarantees*, which provide information that is not explicitly contained in the categories. However, we were able to capture this information implicitly by interpreting the category descriptions. Sometimes, this may also involve investigating other parts of the GT results, for instance, details contained in subcategories or otherwise related categories (cf. Figure 3). Another example for implicit information is the field *Priority*. Although Cat2.4 suggests that the described activities were somehow important but no essential, we mapped this information to a value of an appropriately chosen priority scale. Table 4 shows the example use case after the second pass in the middle column.

In order to complete the content transfer, we reviewed all use cases and assessed whether we had to combine or even split some of them. By this point, we also created a use case diagram. Therefore, it may be useful to resort to underlined references to other use cases. Figure 4 illustrates the use case diagram for the model merging example after the second pass. It also shows that the total amount of about 80 categories could be reduced to less than thirty use cases.

Table 4: Evolution of UC-20 Annotate matchings. Only fields marked with an asterisk (*) are considered during the structure transfer.

	Structure*/Content transfer	**Detail completion**
Use Case: 20	**Name:** Annotate matchings	
Description*	Strategies to record observations on the similarity of two model elements. Marking of two elements of the same type (also groups of elements), marking of distributed features (1:n matchings), and marking of implicit and explicit realizations (semantic matchings).	A user wants to record observations on the similarity of two model elements visually. Here, three types of annotations are common: marking of two elements of the same type (also groups of elements), marking of distributed features (1:n matchings), and marking of implicit and explicit realizations (semantic matchings). The model merge tool (MMT) then annotates user-selected matchings according to those types of similarity either automatically or asks the user for confirmation.
Scope	Annotating the source diagrams	
Level	User	
Primary actor*	Participant/Subject	Model merge tool (MMT)
Stakeholders and interests	Participant wants to record matching information visually. Thus, she can continue her investigations but is still able to resort to this graphically recorded information.	[User and MMT] The MMT supports the user to record matching information visually. Thus, the user can continue her investigations but is still able to resort to this graphically recorded information.
Preconditions*	A matching between two model elements (or groups of elements) has been identified.	Both source diagrams contain selected elements. Either done manually by the user or automatically by the MMT.
Success guarantees	Identified matchings are annotated.	Selected matchings are annotated.
Trigger	Participant wants to record matching information visually.	Existing matching information will be recorded visually.
Main success scenario	1. The participant annotates model elements or groups of elements of the same type based on a certain annotation strategy.	1. User instructs the MMT to annotate a matching that was selected by herself or automatically by the tool in a previous step. 2. MMT annotates the matching according to the three types of similarity.
Extensions or alternatives	1a. The annotation strategy depends on the three types of similarity.	1a. MMT is in auto mode. 1a1. Goto 2.
Associated information*	Cat2.4 Annotating of Matchings and Subcategories Cat2.4.1 Marking of two elements of the same type, Cat2.4.2 Marking of distributed features, Cat2.4.3 Marking of implicit and explicit realizations	
Open issues*		Better term for “auto mode”
Change history*	08/24 (Structure transfer) → 09/18 (Content transfer) → 09/28 (Detail completion)	
Priority	Medium - Subjects that did not pursue this strategy were often forced to reinvestigate previous problems.	High - Subjects that did not pursue this strategy were often forced to reinvestigate previous problems. Thus, this use case may avoid redundancies, and annotations can be important for other features.
Modifications of physical artifacts	• Participants used several colors in combination with different line and shape types: 1. Found matchings were highlighted with the same color in both source diagrams. 2. Colors were used to annotate a matching type and not a specific matching itself. • Pointers to elements of the other diagram and naming of groups of elements with capital letters	
Miscellaneous	• Different annotation strategies depending on the processing habits of the participant and the level of detail. • Annotations may be a basis for detailed investigations of similarity especially when the complexity increases.	• All matching information should be visualized. • The type of similarity is transferable to 1:1, 1:n, and n:m matchings. Thus, different annotation strategies can be chosen and the MMT may determine them automatically.

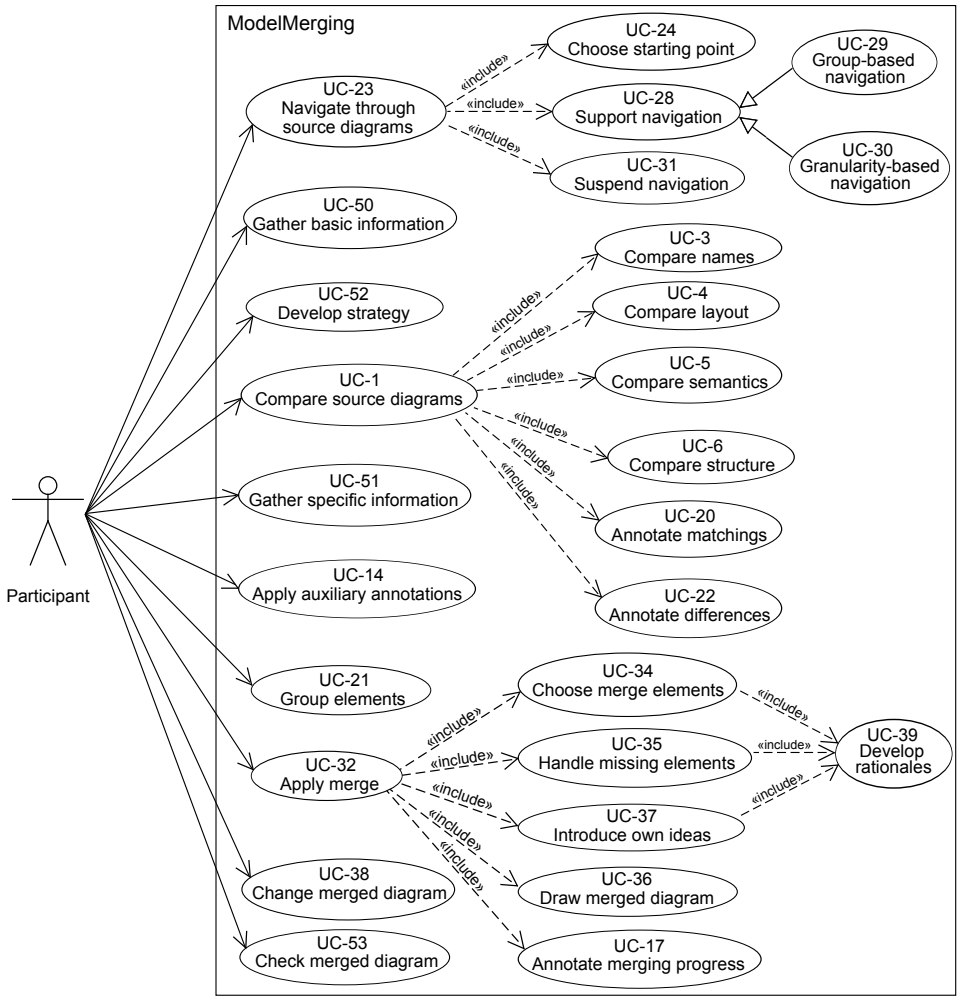

Figure 4: Use case diagram for model merging after the content transfer.

4.2.3 Detail completion

The main goal of the third pass is to translate the observations from the reality to a software perspective, i.e., we had to modify the use cases in such a way that these reflect the software system to be built rather than the activities of a participant in the GT study. Furthermore, open issues should be solved and system specific use cases may be added. Following common use case practice, we provided a name for the software system, here, *Model Merge Tool* (MMT).

The right column of Table 4 shows the outcome of the detail completion pass. Based on our experience, we suggest to start with the fields *Preconditions* and *Main success scenario*. In our example, we made the *Preconditions* more precise by determining when "a matching between two model elements has been identified". Using the MMT, matchings may be identified in two ways: manually by the user or automatically by the tool itself. In either way those elements that are part of a matching should be selected to enable possible annotations. This is also reflected in the first step of the *Main success scenario*. Moreover, the second step shows that we decided to include support for annotations of different types of similarities, which was only mentioned as alternative in the previous version of the use case. This decision may be either made by the GT2UC team itself in case it has sufficient domain knowledge or could also result from a discussion with the stakeholders. While the main success scenario suggests that the user triggers the annotation of a

selected matching manually, a possible *Alternative* is to let the tool do this automatically for those matchings, which were detected by the tool itself (auto mode). In order to determine which type of similarity a certain matching covers, another discussion revealed additional details stored in the *Miscellaneous* field.

Note that the priority of this use case was changed. Although the same reason still applies, we decided to raise the priority from medium to high. Here, we argue that an annotation of matchings shall always be applied in order to avoid redundancies and to achieve a high degree of automation for the MMT. Based on the main success scenario, its alternatives, and the previous description, we were able to adapt the current description of the use case. Furthermore, we changed the *Primary actor* to be the MMT as the user only plays a supportive or minor role.

In this example, translating the remaining fields to the software perspective was rather straightforward. However, some information may become irrelevant, might be replaced as shown for the *Miscellaneous* field, or is directly copied from the previous version of the use case.

All of the other use cases from the second pass are processed accordingly. Even new use cases may be created to model features of the software system that were not covered by the GT study like loading the source diagrams or saving the merged model. Such use cases may require further discussion with domain experts or developers. Furthermore, the structure of some of the use cases will probably change, others might be combined, or split. Thus, more than a single iteration of the detail completion pass might be reasonable. Moreover, we had to adjust the use case diagram, which now depicts the behavior of the software system, here, the MMT.

5 Limitations

Although GRE shows itself to be valuable for our own purposes, we want to discuss some limitations to this method.

Applying GRE may be time consuming not only due to the GT itself, but also because of possible iterations during the GT2UC step. However, we argue that especially novel software systems benefit from complete, comprehensible, and grounded requirements.

The gained requirements heavily depend on a carefully applied GT and a well considered and structured transfer of its results to use cases. At a first glance, this might only shift a possible error source from the customer to the team in charge of GRE. However, assuming that such a team is experienced in applying the GT/GRE this can reduce the overall risk.

In this paper we demonstrated GRE for the task of model merging and only discussed the transfer of a single use case in detail. Eliciting requirements for other applications could help to further refine our method. Although we transferred all categories to the use case model, this process might differ for other applications, there might appear new problems that have to be tackled differently.

As discussed in Section 2.1, researchers that apply the GT must at least have the required contextual knowledge and *theoretical sensitivity*. That is, they must have a certain domain knowledge in order to understand and interpret the data correctly.

Obviously, our approach inherits the limitations of the particular data collection method (e.g. thinking aloud). However, requirements engineers are free to replace its with other methods.

6 Related work

In general, it is not unusual that requirements engineering leverages techniques from cognitive and social sciences to help eliciting and analyzing requirements as a significant part of requirements engineering is concerned with the interpretation of observations or stakeholders' views [NE00]. In this section we briefly discuss papers that used the Grounded Theory for requirements engineering in the field of software engineering or (information) systems engineering in general. However, our extensive literature review revealed that there only exist a few papers which address this particular issue.

Halaweh [Hal11] also recognizes the usefulness of the GT in requirements engineering. He highlights that the GT can leverage systematics in RE and that it can prevent early commitment by helping requirements analysts to remain open. In his paper he briefly proposes the transfer of GT results to entities such as UML class diagrams or entity relationship diagrams. Chakraborty and Dehlinger [CD09] follow a similar approach. They make use of the GT to construct a system architecture from Enterprise System Requirements. Mostly textual requirements are transformed to class diagrams in a straightforward way, e.g., in open coding categories could be directly mapped to classes of UML class diagrams. Berry et al. [BGH+08] even argue that requirements engineering is nothing else than the application of the GT methods "to discover and construct requirements of the computer-based system that a client needs and wants". Results from RE—that is to say requirements representations—are Grounded Theories. Finally, Berry et al. describe how tasks in requirements engineering and, moreover, architecture recovery can be mapped to specific steps of the GT. While the above approaches seem to be straightforward at first, we discovered that the transfer from the GT results to requirements is less obvious and we think that the transfer itself is an important part of the whole process. Moreover, we found that it is crucial to create a more compact set of requirements based the GT results.

From the field of information systems engineering, Galal and Paul [GP99] introduced the grounded systems engineering methodology (GSEM). They propose to use the GT to analyze the current state of the existing system, use the results to develop a prototypical system, and validate it in an iterative approach. They argue that this methodology works especially well in "ill-structured problem situations—those in which the requirements for a solution are not well understood" similar to what we described as unawareness (I1). However, they focus on the complete process itself and less on intermediate results which makes it difficult to compare theirs to our approach.

An explorative application of the GT in the context of process engineering is investigated by Carvalho et al. [CSJ03]. They compare the resulting requirement artifacts of a software engineer using ad-hoc-methods and a psychologist using the GT. The conclusion is that a sole application of the GT cannot replace the knowledge of an experienced software engineer but that "methods such as the grounded theory, in which a model is generated from the data, could possibly provide traceability and a more repetitive approach". When unstructured ad-hoc methods are applied, artifacts sometimes were neglected or forgotten. Unfortunately, the authors do not provide recommendations on how to adopt the GT for RE.

7 Conclusions

In this paper we introduced a systematic approach to get from real-world observations to requirements in form of use cases and illustrated our method by a non-trivial example. Our approach consists of two steps: First, real-world observations are analyzed to produce abstract descriptions of the user behavior. Second, these descriptions are transferred into use cases. Since, we use the GT for the first step, we put more emphasis on discussing and illustrating the second step in this paper. We argue that our approach addresses the three issues mentioned in the introduction: Early commitment is prevented by observing users solving real-world tasks without using or thinking about using specialized software. As the GT strives to reach theoretical saturation it should uncover all requirements that can be derived from the observed user behavior. Finally, both steps (GT and GT2UC) involve linking results like categories or use cases to their underlying observations. By addressing these issues GRE fills a gap in the arsenal of elicitation methods of today's requirement engineers.

Although not yet used in industry, we have extensive documentation and illustrative examples and, moreover, we applied GRE to a non-trivial software project. We are eager to find industry partners to see GRE in practice.

References

[All03] George Allan. A Critique of using Grounded Theory as a Research Method. *Electronic Journal of Business Research Methods*, 2(1):1–10, July 2003.

[BGH+08] Daniel M. Berry, Michael W. Godfrey, Ric Holt, Cory J. Kapser, and Isabel Ramos. Requirements Specifications and Recovered Architectures as Grounded Theories, 2008.

[Boe81] Barry W. Boehm. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[BS08] Andrew Begel and Beth Simon. Novice software developers, all over again. In *Proc. of the Workshop on Computing Education Research, Sydney, Australia, Sept. 6-7*, pages 3–14. ACM, 2008.

[CA07] Betty H. C. Cheng and Joanne M. Atlee. Research Directions in Requirements Engineering. In *Proc. of the International Conference on Software Engineering, Workshop on the Future of Software Engineering, Minneapolis, MN, USA, May 23-25*, pages 285–303. IEEE Computer Society, 2007.

[CD09] Suranjan Chakraborty and Josh Dehlinger. Applying the Grounded Theory Method to Derive Enterprise System Requirements. In *Proc. of the International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*, pages 333–338. IEEE Computer Society, 2009.

[Coc00] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley Longman, 2000.

[CSJ03] Lucila Carvalho, Louise Scott, and Ross Jeffery. Exploring the use of Techniques from Grounded Theory in Process Engineering. Technical report, Centre for Advanced Software Engineering Research, 2003.

[CSN09] Carlton A. Crabtree, Carolyn B. Seaman, and Anthony F. Norcio. Exploring language in software process elicitation: A grounded theory approach. In *Proc. of the International Symposium on Empirical Software Engineering and Measurement, October 15-16, 2009, Lake Buena Vista, Florida, USA*, pages 324–335. IEEE Computer Society, 2009.

[Dav05] Alan M. Davis. *Just Enough Requirements Management: Where Software Development Meets Marketing*. Dorset House Publishing Company, New York, 2005.

[DSTH12] Laura A. Dabbish, H. Colleen Stuart, Jason Tsay, and James D. Herbsleb. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proc. of the Conference on Computer Supported Cooperative Work, Seattle, WA, USA, Feb. 11-15, 2012*, pages 1277–1286, 2012.

[ES80] K. Ericsson and H. Simon. Verbal reports as data. *Psychological Review*, 87(3):215–251, 1980.

[FBC11] Dominic Furniss, Ann Blandford, and Paul Curzon. Confessions from a grounded theory PhD: experiences and lessons learnt. In *Proc. of the Conference on Human Factors in Computing Systems, Vancouver, BC, Canada, May 7-12*, pages 113–122. ACM, 2011.

[GF94] Orlena C. Z. Gotel and Anthony C. W. Finkelstein. An Analysis of the Requirements Traceability Problem. In *Proc. of the International Conference on Requirements Engineering, Colorado Springs, CO, USA, Apr. 18-22*, pages 94–101. IEEE, 1994.

[GP99] Galal Hassan Galal and Ray J. Paul. A Qualitative Scenario Approach to Managing Evolving Requirements. *Requirements Engineering*, 4:92–102, 1999.

[GS67] Bernie Glaser and Anselm L. Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine, Chicago, 1967.

[Hal11] Mohanad Halaweh. Using Grounded Theory as a Supportive Technique for System Requirements Analysis. In *Proc. of the International Conference on Systems, St. Maarten, The Netherlands Antilles, Jan. 23-28*, pages 54–59, 2011.

[HNM10] Rashina Hoda, James Noble, and Stuart Marshall. Using grounded theory to study the human aspects of software engineering. In *Proc. of the Workshop on Human Aspects of Software Engineering, Reno/Tahoe, Nevada, USA, October 17-21*, pages 5:1–5:2. ACM, 2010.

[IEE90] IEEE Standard Glossary of Software Engineering Terminology, 1990.

[KG00] Daryl Kulak and Eamonn Guiney. *Use Cases: Requirements in Context*. Addison-Wesley Longman, 2000.

[LWD11] Rainer Lutz, David Würfel, and Stephan Diehl. How Humans Merge UML-Models. In *Proc. of the International Symposium on Empirical Software Engineering and Measurement, Banff, Alberta, Canada, Sept. 22-23*, pages 177–186. IEEE, 2011.

[NE00] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *Proc. of the International Conference on Software Engineering, Future of Software Engineering Track, Limerick, Ireland, June 4-11*, pages 35–46. ACM, 2000.

[Nie93] J. Nielsen. *Usability Engineering*. Academic Press, 1993.

[NK91] Takeshi Nakajo and Hitoshi Kume. A Case History Analysis of Software Error Cause-Effect Relationships. *IEEE Trans. Software Eng.*, 17(8):830–838, 1991.

[SC08] Anselm L. Strauss and Juliett Corbin. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications, Thousand Oaks, CA, 2008.

A Categories (in German)

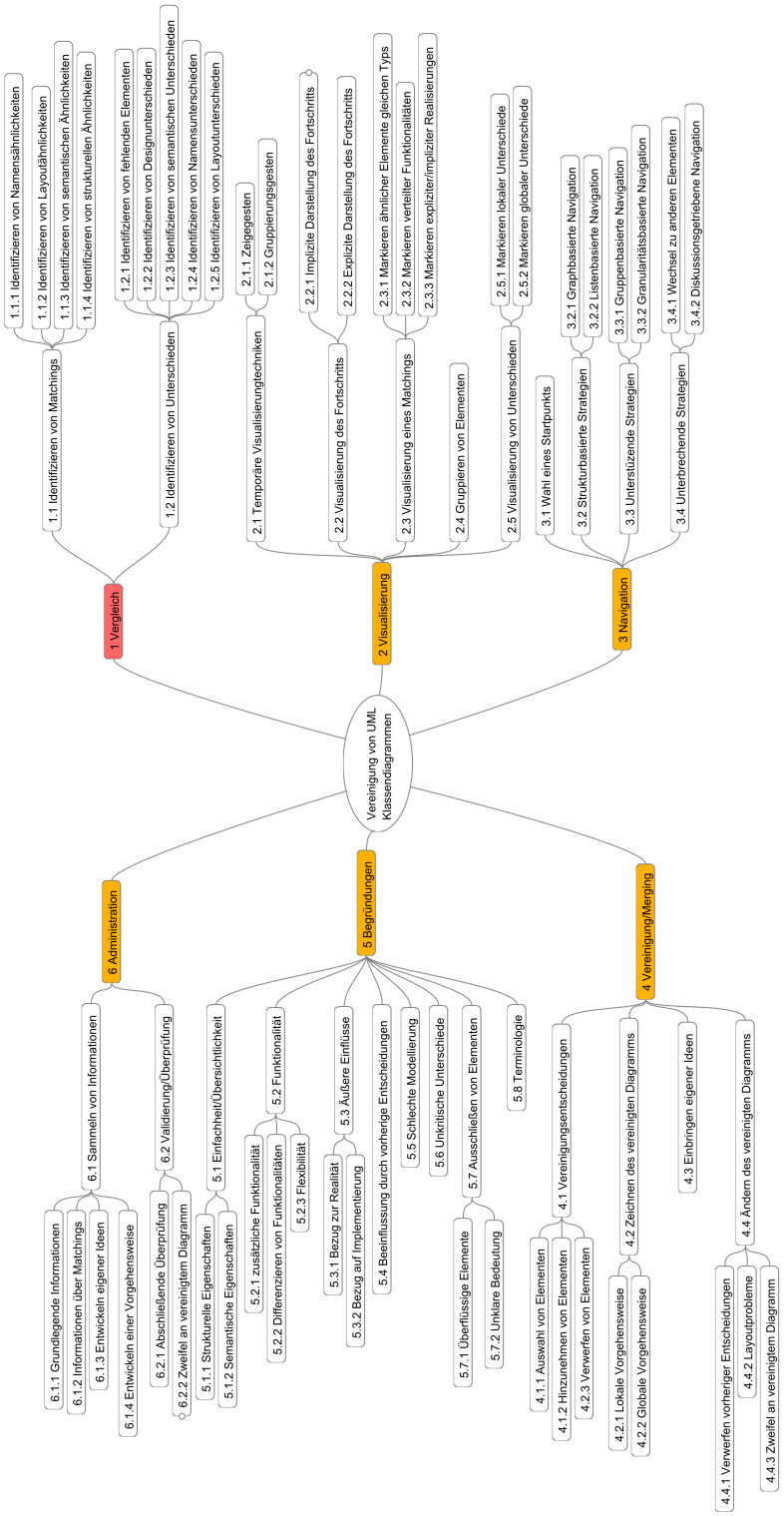

Figure 5: Categories and their relations

B Use Cases (in German) — First Iteration (Structure Transfer)

Use Case: UC-1	**Name:** Ausgangsdiagramme vergleichen
Kurzbeschreibung:	Vergleich der Ausgangsdiagramme. Hierin werden sowohl Ähnlichkeiten als auch Unterschiede identifiziert. Probanden arbeiten sich häufig von globaler zu lokaler Betrachtungsweise (Granularität von grob nach fein) vor. Analyse/Beurteilung der Ähnlichkeit beider Ausgangsdiagramme und der darin enthaltenen Elemente. Ziel ist es häufig (wenn auch teilweise unbewusst), eine Grundmenge/ein Grundgerüst von Matchings zu identifizieren. Existiert ein Element nur in einem der Diagramme, d.h. für dieses lässt sich kein Pendant im anderen Ausgangsdiagramm finden, so wird dieses als fehlendes Element identifiziert. Analyse/ Beurteilung der Unterschiede zwischen den Ausgangsdiagrammen und den darin enthaltenen Elementen. Kann kein Matching mehr gefunden werden, so ergibt sich aus den übrigen Elementen der Unterschied. Suchen von Unterschieden wird lokal auf diese Matchings beschränkt.
Primärer Akteur:	Proband
Vorbedingungen:	- Grundlegende Informationen über Diagramme und die Domäne bereits bekannt. - Navigation innerhalb der Diagramme notwendig.
Dazugehörige Informationen:	Kategorie 1 Vergleich, 1.1 Matchings identifizieren, 1.2 Unterschiede identifizieren, 1.2.1 Identifizieren fehlender Elemente

Use Case: UC-2	**Name:** Matchings identifizieren
Kurzbeschreibung:	Analyse/Beurteilung der Ähnlichkeit beider Ausgangsdiagramme und der darin enthaltenen Elemente. Ziel ist es häufig (wenn auch teilweise unbewusst), eine Grundmenge/ein Grundgerüst von Matchings zu identifizieren. Existiert ein Element nur in einem der Diagramme, d.h. für dieses lässt sich kein Pendant im anderen Ausgangsdiagramm finden, so wird dieses als fehlendes Element identifiziert.
Primärer Akteur:	Proband
Vorbedingungen:	- Grundlegende Informationen über Diagramme und die Domäne bereits bekannt. - Navigation innerhalb der Diagramme notwendig.
Dazugehörige Informationen:	Kategorie 1.1 Identifizieren von Matchings, 1.2.1 Identifizieren fehlender Elemente

Use Case: UC-3	**Name:** Namen vergleichen
Kurzbeschreibung:	Vergleich von Namen/Bezeichnern zum Identifizieren ähnlicher Elemente, um ein grundlegendes Matching der beiden Ausgangsdiagramme zu erreichen, und Identifizieren semantischer Unterschiede. Funktionalität, Aufgabe und Zweck ähnlicher Elemente werden verglichen und semantische Unterschiede herausgearbeitet.
Primärer Akteur:	Proband
Dazugehörige Informationen:	1.1.1 Identifizieren von Namensähnlichkeiten, 1.2.3 Identifizieren von sem. Unterschieden

Use Case: UC-4	**Name:** Layout vergleichen
Kurzbeschreibung:	Ähnlichkeiten im Layout beobachten. Ziel ist es, ein Matchings über das Layout zu identifizieren. Werden Layoutähnlichkeiten identifiziert, so könnten daher Namensunterschiede auftreten. Werden keine Ähnlichkeiten im Layout entdeckt, so werden diese als Layoutunterschiede identifiziert. Funktionalität, Aufgabe und Zweck ähnlicher Elemente werden verglichen und semantische Unterschiede herausgearbeitet.
Primärer Akteur:	Proband
Dazugehörige Informationen:	1.1.2 Identifizieren von Layoutähnlichkeiten, 1.2.4 Identifizieren von Namensunterschieden, 1.2.5 Identifizieren von Layoutunterschieden, 1.2.3 Identifizieren von sem. Unterschieden

Use Case: UC-5	**Name:** Semantik vergleichen
Kurzbeschreibung:	Strategie zum Identifizieren von Matchings. Meist wird die Funktionalität, die Aufgabe oder der Zweck eines Elements mit der/dem eines anderen verglichen. Empfinden Probanden diese(n) als hinreichend ähnlich, so behandeln sie die gefundenen Elemente als Matching. Werden semantische Ähnlichkeiten identifiziert, so können daher Namensunterschiede auftreten. Weiterhin können Designunterschiede identifiziert werden. Ziel ist dabei, das Matching von semantischen Ähnlichkeiten bezüglich Realisierung in beiden Ausgangsdiagrammen zu untersuchen und über die gefundenen Unterschiede zwischen Vor- und Nachteilen der jeweiligen Modellierung abzuwägen. Art des gefundenen Unterschieds festhalten.

Primärer Akteur:	Proband
Dazugehörige Informationen:	Kategorie 1.1.3 Identifizieren von sem. Ähnlichkeiten, 1.2.4 Identifizieren von Namensunterschieden, 1.2.2 Identifizieren von Designunterschieden

Use Case: UC-6	**Name:** Struktur vergleichen
Kurzbeschreibung:	Strukturelle Ähnlichkeiten beschreiben die Anbindung an andere Elemente im Diagramm, was in diesem Fall das ausschlaggebende Merkmal für das Identifizieren von Matchings ist. Werden strukturelle Ähnlichkeiten identifiziert, so sind Namensunterschiede vorhanden. Funktionalität, Aufgabe und Zweck ähnlicher Elemente werden verglichen und semantische Unterschiede herausgearbeitet.
Primärer Akteur:	Proband
Vorbedingung	Meist, wenn Namensähnlichkeiten nicht ausreichen, um ein Matching festzulegen.
Dazugehörige Informationen:	Kategorie 1.1.4 Identifizieren von struk. Ähnlichkeiten, 1.2.4 Identifizieren von Namensunterschieden, 1.2.3 Identifizieren von sem. Unterschieden

Use Case: UC-8	**Name:** Fehlende Elemente identifizieren
Kurzbeschreibung:	Ein Element existiert nur in einem der Diagramme, d.h. für dieses lässt sich kein Pendant im anderen Ausgangsdiagramm finden.
Primärer Akteur:	Proband
Vorbedingungen:	Bleiben nach dem Identifizieren von Matchings übrig.
Dazugehörige Informationen:	Kategorie 1.2.1 Identifizieren fehlender Elemente

Use Case: UC-12	**Name:** Layoutunterschiede identifizieren
Kurzbeschreibung:	Implizit beim Suchen von Ähnlichkeiten im Layout entdeckt.
Primärer Akteur:	Proband
Vorbedingungen:	Keine Ähnlichkeiten im Layout entdeckt.
Dazugehörige Informationen:	Kategorie 1.2.5 Identifizieren von Layout ähnlich

Use Case: UC-14	**Name:** Temporäre Visualisierungstechniken anwenden
Kurzbeschreibung:	Temporäre Visualisierungstechniken beim Sammeln grundlegender Informationen, Überprüfen der Ähnlichkeit zweier Klassen und teilweise dem Erweitern des vereinigten Diagramms. Fokussieren bestimmter Elemente innerhalb der Ausgangsdiagramme. Auf ein Element mit jeweils einer Hand zeigen, wobei Ziel ist, bestimmte Bereiche bzw. Elemente innerhalb der Ausgangsdiagramme optisch festzuhalten. Zeigegesten haben eine Art Filtercharakter und helfen dem Probanden, sich auf das aktuelle Problem zu konzentrieren. Gruppierungsgesten fassen eine Menge von Elementen temporär zu einer Gruppe zusammen.
Primärer Akteur:	Proband
Dazugehörige Informationen:	Kategorie 2.1 Temporäre Visualisierungstechniken, 2.1.1 Zeigegesten, 2.1.2 Gruppierungsgesten

Use Case: UC-17	**Name:** Mergefortschritt visualisieren
Kurzbeschreibung:	Visualisierung des Fortschritts im Vereinigungsprozess. Verschiedene Techniken zur Markierung von Modellelementen innerhalb der Ausgangsdiagramme. Ziel ist es, Modellelemente zu identifizieren, welche noch nicht behandelt wurden und somit die Navigation(smöglichkeiten) innerhalb der Ausgangsdiagramme zu beschränken. Zum einen implizite Darstellung des Fortschritts durch Markieren sämtlicher bereits behandelter Elemente, wobei Matchings an beiden Stellen markiert werden. Weiterhin Streichen verworfener Elemente, Markieren übernommener Elemente. Zum anderen explizite Darstellung des Fortschritts durch Markieren noch zu behandelnder Elemente. Einmalig vor allem gegen Ende des Vereinigungsprozesses und in Kombination auf implizite Darstellung folgend.
Primärer Akteur:	Proband
Vorbedingungen:	Getroffene Vereinigungsentscheidungen und Überführen dieses Ergebnisses in das vereinigte Diagramm.
Dazugehörige Informationen:	Kategorie 2.2 Visualisierung des Fortschritts, 2.2.1 Implizite Darstellung des Fortschritts, 2.2.1.1 Markieren abgearbeiteter Elemente, 2.2.1.2 Streichen verworfener Elemente, 2.2.1.3 Markieren übernommener Elemente, 2.2.2 Explizite Darstellung des Fortschritts

Use Case: UC-20	**Name:** Matching visualisieren
Kurzbeschreibung:	Strategien zum Aufzeichnen/Festhalten von Beobachtungen bezüglich der Ähnlichkeit zweier Elemente. Unterschiedliche Markierungsstrategien. Markieren zweier Elemente gleichen Typs. Gruppen von Elementen als ähnlich markieren. Markieren verteilter Funktionalitäten (Gruppen). Markieren expliziter/impliziter Realisierungen (semantisches Matching).
Primärer Akteur:	Proband
Vorbedingungen:	Matching zwischen zwei Elementen identifiziert.
Dazugehörige Informationen:	Kategorie 2.3 Visualisierung eines Matchings, 2.3.1 Markieren ähnlicher Elemente gleichen Typs, 2.3.2 Markieren verteilter Funktionalitäten, 2.3.3 Markieren expl./impl. Realisierungen

Use Case: UC-21	**Name:** Elemente gruppieren
Kurzbeschreibung:	Definieren von Elementgruppen innerhalb eines Diagramms zur Vorbereitung auf das eigentliche Markieren ähnlicher Elemente und zur Erleichterung des Vereinigungsproblems.
Primärer Akteur:	Proband
Dazugehörige Informationen:	Kategorie 2.4 Gruppieren von Elementen

Use Case: UC-22	**Name:** Unterschiede visualisieren
Kurzbeschreibung:	Visualisierung von Unterschieden. Markieren lokaler Unterschiede (Attribute, Methoden, Kardinalitäten) und globaler Unterschiede (Klassen, Strukturen/Konstrukte). Ziel ist es, Informationen für eine spätere Betrachtung bzw. Erzeugung des vereinigten Diagramms in einem späteren Schritt festzuhalten.
Primärer Akteur:	Proband
Vorbedingungen:	Identifizierte/markierte Matchings (lokal). Fehlende Elemente identifiziert (global).
Dazugehörige Informationen:	Kategorie 2.5 Visualisierung von Unterschieden, 2.5.1 Markieren lokaler Unterschiede, 2.5.2 Markieren globaler Unterschiede

Use Case: UC-23	**Name:** Durch Ausgangsdiagramme navigieren
Kurzbeschreibung:	Strategien, die zur Navigation innerhalb der Ausgangsdiagramme verwendet werden. Muss vor allem einzeln durch die Diagramme navigiert werden. Auch synchronisiertes Navigieren, d.h. die entsprechenden Navigationstechniken müssen u.U. auf beide Diagramme angewendet werden. Durchlaufen der Diagramme durch Ausnutzen des zugrundeliegenden Graphen. Ausgehend vom gerade betrachteten Element wird über dessen Beziehungen das nächste Element bestimmt und anschließend analysiert. Ist ein Startpunkt gefunden, so werden Ausgangsdiagramme häufig in einer der Tiefensuche ähnlichen Vorgehensweise durchlaufen. Wird ein Diagramm bevorzugt, so liegt der Fokus häufig auf diesem. I.A. wenn kein bevorzugtes Diagramm existiert, kann der Fokus auch zwischen den Diagrammen wechseln. Klassenintern wird der Inhalt der betrachteten Klassen meist (parallel) von oben nach unten abgearbeitet.
Primärer Akteur:	Proband
Vorbedingungen:	Graphbasiert: Startpunkt gefunden. Listenbasiert: Matching identifiziert, wenn Navigation in beiden Ausgangsdiagrammen durchgeführt werden soll.
Dazugehörige Informationen:	Kategorie 3 Navigation, 3.2 Strukturbasierte Strategien, 3.2.1 Graphbasierte Navigation, 3.2.2 Listenbasierte Navigation

Use Case: UC-24	**Name:** Startpunkt wählen
Kurzbeschreibung:	Wählen eines Startpunkts (zu Beginn der Vergleichs- und/oder Vereinigungsphase). Legt fest, welches Element/Matching als erstes betrachtet bzw. in das vereinigte Diagramm überführt werden soll.
Primärer Akteur:	Proband
Dazugehörige Informationen:	Kategorie 3.1 Wahl eines Startpunkts

Use Case: UC-28	**Name:** Navigation unterstützen
Kurzbeschreibung:	Unterstützen strukturbasierter Navigation, indem diese vereinfacht und Komplexität verringert wird.
Primärer Akteur:	Proband
Dazugehörige Informationen:	Kategorie 3.3 Unterstützende Strategien

Use Case: UC-29	**Name:** Gruppenbasiert navigieren
Kurzbeschreibung:	Divide and Conquer Strategie, sodass man jede Gruppe separat abarbeitet und gegebenfalls Beziehungen zu anderen Gruppen betrachtet. Wahl der nächsten Gruppe graphbasiert oder unabhängig (Nummerierung, Leserichtung).
Primärer Akteur:	Proband
Vorbedingungen:	Gruppierung von Elementen
Dazugehörige Informationen:	Kategorie 3.3.1 Gruppenbasierte Navigation

Use Case: UC-30	**Name:** Granularitätsbasiert navigieren
Kurzbeschreibung:	Hier können Klassen selbst, klasseninterne Details oder Beziehungen zwischen Klassen separat betrachtet werden. Wechsel der Betrachtungsebene bedeutet, dass Ausgangsdiagramme erneut durchlaufen werden, allerdings mit anderem Kontext.
Primärer Akteur:	Proband
Vorbedingungen:	Gedankliches Einteilen der Diagramme zu Beginn der Vergleichsphase in Betrachtungsebenen.
Dazugehörige Informationen:	Kategorie 3.3.2 Granularitätsbasierte Navigation
Offene Punkte:	- Wann geschieht ein Auswählen der Betrachtungsebene?

Use Case: UC-31	**Name:** Navigation unterbrechen
Kurzbeschreibung:	Es wird nicht selten zu einem einfacheren oder unabhängigen Element/Problem gewechselt und das aktuelle zurückgestellt. Sind die neuen Probleme gelöst, so wird meist direkt oder über verwandte Probleme zum ursprünglichen Problem zurückgekehrt.
Primärer Akteur:	Proband
Vorbedingungen:	Strukturbasierte Strategie wird verfolgt.
Dazugehörige Informationen:	Kategorie 3.4 Unterbrechende Strategien, 3.4.1 Wechsel zu anderen Elementen

Use Case: UC-32	**Name:** Merge durchführen
Kurzbeschreibung:	Techniken zum Durchführen der gleichnamigen Vereinigungsphase. Es müssen Entscheidungen bezüglich des vereinigten Diagramms in der Vereinigungsphase getroffen werden. Ist die Entscheidung getroffen, so wird (wenn nötig) das vereinigte Diagramm entsprechend angepasst.
Primärer Akteur:	Proband
Vorbedingung:	Basiert auf zuvor gefundenen Matchings sowie erarbeiteten Begründungen.
Dazugehörige Informationen:	Kategorie 4 Vereinigung/Merging, 4.1 Vereinigungsentscheidungen

Use Case: UC-34	**Name:** Mergeelemente auswählen
Kurzbeschreibung:	Auswählen eines der zu einem Matching gehörenden Elemente und anschließendes Einfügen dieses Elements in das vereinigte Diagramm. Implizites Verwerfen des anderen Elements. Hinzugefügt durch Zeichnen des vereinigten Diagramms.
Primärer Akteur:	Proband
Vorbedingung:	Ähnliche Elemente in den Ausgangsdiagrammen identifiziert.
Dazugehörige Informationen:	Kategorie 4.1.1 Auswahl von Elementen

Use Case: UC-35	**Name:** Fehlende Elemente behandeln
Kurzbeschreibung:	Verwerfen oder Ergänzen des vereinigten Diagramms durch Elemente, die nur in einem der Ausgangsdiagramme vorhanden sind. Wie das Element im vereinigten Diagramm ergänzt wird, beschreibt die Kategorie 'Zeichnen des vereinigten Diagramms'.
Primärer Akteur:	Proband
Vorbedingung:	Unterschiede gefunden und geeignete Begründung entwickelt.
Dazugehörige Informationen:	Kategorie 4.1.2 Hinzunehmen von Elementen, 4.1.3 Verwerfen von Elementen

Use Case: UC-36	**Name:** Vereinigtes Diagramm zeichnen
Kurzbeschreibung:	Beschreibt den Prozess des Zeichnens eines UML Klassendiagramms, also dem Hinzufügen von Designelementen zum vereinigten Diagramm während der Vereinigungsphase. Es kann zwischen lokaler und globaler Vorgehensweise unterschieden werden.

Primärer Akteur:	Proband
Vorbedingung:	Basiert auf zuvor erarbeiteten Vereinigungsentscheidungen.
Nachbedingung:	Betrachtung des jeweiligen Matchings abgeschlossen.
Dazugehörige Informationen:	Kategorie 4.2 Zeichnen des vereinigten Diagramms, 4.2.1 Lokale Vorgehensweise, 4.2.2 Globale Vorgehensweise
Offene Punkte:	Wie lokale/globale Vorgehensweise geeignet einfügen (auf zu hohem, abstraktem Level)?

Use Case: UC-37	**Name:** Eigene Ideen einbringen
Kurzbeschreibung:	Zum Erhalten von Funktionalität, Semantik und/oder Validität, Ausgangsdiagramme um eigene Ideen ergänzen oder Identifizieren zusätzlicher Funktionalität. Jede beliebige Art der Anpassung.
Primärer Akteur:	Proband
Dazugehörige Informationen:	Kategorie 4.3 Einbringen eigener Ideen, 6.1.3 Entwickeln eigener Ideen

Use Case: UC-38	**Name:** Vereinigtes Diagramm ändern
Kurzbeschreibung:	Bezüglich der Vereinigung getroffene Entscheidungen können sich als schlecht oder fehlerhaft herausstellen. Nötig, das vereinigte Diagramm entsprechend anzupassen und vorherige Entscheidungen zu verwerfen.
Primärer Akteur:	Proband
Dazugehörige Informationen:	Kategorie 4.4 Ändern des vereinigten Diagramms, 4.4.1 Verwerfen vorheriger Entscheidungen, 4.4.2 Layoutprobleme, 4.4.3 Zweifel an vereinigtem Diagramm

Use Case: UC-39	**Name:** Begründungen entwickeln
Kurzbeschreibung:	Begründungen, die Probanden während des Gesamtprozesses entwickeln. Im Allgemeinen beruhen Begründungen auf der Bewertung des Designs der Ausgangsdiagramme. Es können unterschiedliche Entscheidungen miteinander kombiniert werden oder sich gegenseitig beeinflussen. Es können unterschiedliche Gründe für dasselbe Ergebnis existieren.
Primärer Akteur:	Proband
Vorbedingung:	Auf Basis zuvor identifizierter Matchings
Dazugehörige Informationen:	Kategorie 5 Begründungen

Use Case: UC-40	**Name:** Einfachheit bevorzugen
Kurzbeschreibung:	Einfache Modellierung des gegebenen Sachverhalts bevorzugen, sofern diese die geforderte Funktionalität besitzt. Lösung gesucht, welche die gegebenen Anforderungen möglichst gut widerspiegelt. Kann sich auf strukturelle oder semantische Eigenschaften beziehen.
Primärer Akteur:	Proband
Dazugehörige Informationen:	Kategorie 5.1 Einfachheit/Übersichtlichkeit, 5.1.1 Strukturelle Eigenschaften, 5.1.2 Semantische Eigenschaften

Use Case: UC-41	**Name:** Funktionalität erhalten
Kurzbeschreibung:	Beschreibt alle Entscheidungen, die bezüglich der Funktionalität des zu modellierenden Diagramms getroffen werden. Entscheidungen bezüglich zusätzlicher Funktionalität, der Differenzierung von Funktionalität oder der Flexibilität (Erweiterbarkeit/Wiederverwendbarkeit).
Primärer Akteur:	Proband
Dazugehörige Informationen:	Kategorie 5.2 Funktionalität, 5.2.1 Zusätzliche Funktionalität, 5.2.2 Differenzieren von Funktionalität, 5.2.3 Flexibilität

Use Case: UC-42	**Name:** Äußere Einflüsse berücksichtigen
Kurzbeschreibung:	Gründe, die sich nicht ausschließlich auf die Ausgangsdiagramme, deren Modellierung und die dargestellten Funktionalitäten beschränken, sondern durch Erfahrungen der Probanden mit der Domäne selbst erarbeitet werden. Man kann grob zwischen zwei Arten von Einflüssen unterscheiden: Jeglicher Bezug zur Realität und jeglicher Bezug zur Implementierung.
Primärer Akteur:	Proband
Dazugehörige Informationen:	Kategorie 5.3 Äußere Einflüsse, 5.3.1 Bezug zur Realität, 5.3.2 Bezug auf Implementierung

Use Case: UC-43	**Name:** Vorherige Entscheidungen berücksichtigen
Kurzbeschreibung:	Elemente können aufgrund zuvor getätigter Design-/Vereinigungsentscheidungen in das vereinigte Diagramm übernommen werden. Elemente, die aus dem gleichen Diagramm kommen oder einer der Varianten aufgrund spezieller Modellierung.
Primärer Akteur:	Proband
Vorbedingung:	Aufgrund zuvor getätigter Design-/Vereinigungsentscheidungen.
Dazugehörige Informationen:	Kategorie 5.4 Beeinflussung durch vorh. Entscheidungen

Use Case: UC-44	**Name:** Schlechte Modellierung vermeiden
Kurzbeschreibung:	Probanden verwerfen die fehlerhaften Varianten, um so eine korrekte Lösung zu erzeugen oder eigene Ideen mit einzubringen.
Primärer Akteur:	Proband
Vorbedingung:	Basiert auf Beobachtungen bezüglich Schwächen/Fehlern in der Modellierung oder des Layouts.
Dazugehörige Informationen:	Kategorie 5.5 Schlechte Modellierung

Use Case: UC-45	**Name:** Unkritische Unterschiede berücksichtigen
Kurzbeschreibung:	Probanden sehen gewisse Unterschiede als nicht kritisch an und schätzen deshalb eine detaillierte Betrachtung als unnötig ein und wählen die naheliegendere, einfachere, allgemeinere oder gar zuletzt/zuerst betrachtete Lösung.
Primärer Akteur:	Proband
Dazugehörige Informationen:	Kategorie 5.6 Unkritische Unterschiede

Use Case: UC-46	**Name:** Elemente ausschließen
Kurzbeschreibung:	Gründe für das Ausschließen von Elementen aus dem vereinigten Diagramm bzw. dem Verwerfen von Elementen der Ausgangsdiagramme. Im Allgemeinen wird zwischen solchen Elementen unterschieden, deren Funktionalität nachvollziehbar ist, welche allerdings als überflüssig angesehen werden und solchen, deren Bedeutung unklar ist.
Primärer Akteur:	Proband
Dazugehörige Informationen:	5.7 Ausschließen von Elementen, 5.7.1 Überflüssige Elemente, 5.7.2 Unklare Bedeutung

Use Case: UC-47	**Name:** Terminologie erhalten
Kurzbeschreibung:	Entscheidungen aufgrund des verwendeten Bezeichners bzw. der verwendeten Terminologie. Aussagekräftigere Varianten bevorzugt, Vermeiden von Sonderzeichen, Beibehaltung der Sprache und Einhaltung von Programmierrichtlinien.
Primärer Akteur:	Proband
Vorbedingung:	Funktionalität von gematchten Elementen ähnlich oder gar identisch.
Dazugehörige Informationen:	Kategorie 5.8 Terminologie

Use Case: UC-50	**Name:** Grundlegende Informationen sammeln
Kurzbeschreibung:	Sammeln grundlegender Informationen über die beiden Ausgangsdiagramme und die damit verbundene Domäne. Sich einen Überblick verschaffen.
Primärer Akteur:	Proband
Dazugehörige Informationen:	Kategorie 6.1 Administration, 6.1.1 Sammeln grundlegender Informationen

Use Case: UC-51	**Name:** Spezifische Informationen sammeln
Kurzbeschreibung:	Spezifische Informationen über Matchings sammeln. Ziel ist es, neben Feinheiten der Modellierung, Unterschieden in der Semantik sowie den daraus resultierenden Vor- und Nachteilen auch eine globale Bewertung der Diagramme zu entwickeln.
Primärer Akteur:	Proband
Dazugehörige Informationen:	Kategorie 6.1 Administration, 6.1.2 Informationen über Matchings

Use Case: UC-52	**Name:** Vorgehensweise entwickeln
Kurzbeschreibung:	Probanden entwickeln unterschiedliche Vorgehensweisen, mit denen sie sich den aufkommenden Problemen stellen. Probanden entwickeln ihre Strategie, während sie sich mit der gestellten Aufgabe beschäftigen; teilweise legen sich Probanden aber auch bewusst eine Strategie bereit bzw. entwickeln diese nach einer ersten globalen Betrachtung der Ausgangsdiagramme.
Primärer Akteur:	Proband
Dazugehörige Informationen:	Kategorie 6.1.4 Entwickeln einer Vorgehensweise

Use Case: UC-53	**Name:** Mergediagramm überprüfen
Kurzbeschreibung:	Innerhalb der Ausgangsdiagramme überprüfen die Probanden abschließend, ob alle dort aufgeführten Elemente abgearbeitet und entsprechend im vereinigten Diagramm realisiert oder ggf. verworfen wurden. Seltener wird abschließend noch einmal das vereinigte Diagramm auf Korrektheit bezüglich Semantik überprüft (Validierung).
Primärer Akteur:	Proband
Dazugehörige Informationen:	Kategorie 6.2 Validierung/Überprüfung, 6.2.1 Abschließende Überprüfung, 6.2.2 Zweifel an vereinigtem Diagramm, 6.2.2.1 Eigene Entscheidungen, 6.2.2.2 Vorherige Entscheidungen

C Use Cases (in German) — Second Iteration (Content Transfer)

Use Case: UC-1	**Name:** Ausgangsdiagramme vergleichen
Kurzbeschreibung:	Vergleich der Ausgangsdiagramme. Identifikation/Analyse/Beurteilung der Ähnlichkeit und Unterschiede beider Ausgangsdiagramme und der darin enthaltenen Elemente. Ziel ist es häufig, eine Grundmenge von Matchings zu identifizieren. Existiert ein Element nur in einem der Diagramme, so wird dieses als fehlendes Element identifiziert. Kann kein Matching mehr gefunden werden, so ergibt sich aus den übrigen Elementen der Unterschied.
Anwendungsbereich:	Analyse der Ausgangsdiagramme
Ebene:	Zusammenfassung
Primärer Akteur:	Proband
Projektbetroffene u. Interessen:	Proband möchte Matchings und Unterschiede finden und visualisieren. Dies ist eine Voraussetzung für den späteren Merge der Ausgangsdiagramme.
Vorbedingungen:	- Ausgangsdiagramme liegen vor. - Grundlegende Informationen über Diagramme und die Domäne bereits bekannt. - Navigation innerhalb der Diagramme notwendig.
Erfolgsgarantien:	Matchings und/oder Unterschiede auf gegebener Granularität wurden identifiziert und visualisiert. Gibt es fehlende Elemente, so wurden diese identifiziert.
Auslöser:	Proband vergleicht Ausgangsdiagramme.
Hauptszenario:	1. Folgende Schritte können parallel und in beliebiger Reihenfolge stattfinden: - Proband vergleicht Namen. - Proband vergleicht Layout. - Proband vergleicht Semantik. - Proband vergleicht Struktur. 2. Proband identifiziert fehlende Elemente. 3. Proband visualisiert die Matchings. 4. Proband visualisiert die Unterschiede.
Erweiterungen/Alternativen:	2a. Es existieren keine Elemente, die nur in einem der beiden Ausgangsdiagramme vorkommen. 2a1. Es gibt keine fehlenden Elemente.
Dazugehörige Informationen:	Kategorie 1 Vergleich, 1.1 Matchings identifizieren, 1.2 Unterschiede identifizieren, 1.2.1 Identifizieren fehlender Elemente
Offene Punkte:	- Parameter 'Granularität'?
Priorität:	Hoch (essentieller Bestandteil des Gesamtprozesses; wichtiger Prozess)
Sonstiges, Anmerkungen:	- Ablauf von global nach lokal (Granularität von grob nach fein)

	- Nur anfangs ein zusammenhängender Prozess, denn auch während dem Erstellen des Mergediagramms werden weitere Vergleiche durchgeführt - Wichtig ist erster Eindruck, den Probanden bei einem ersten globalen Vergleich erlangen. Kann weitere Vorgehensweise prägen - Finden von Matchings bildet Grundlage für weitere detaillierte Analysen (fehlende Elemente, Unterschiede, Visualisierung, Vereinigung) - Matchingstrategien lassen sich zu unterschiedlichen Zeitpunkten anwenden und sind miteinander kombinierbar - Finden von Unterschieden meist implizit beim Identifizieren von Matchings, d.h. Suchen von Unterschieden wird lokal auf Matchings beschränkt - Unterschiede werden während des Merge näher untersucht - Unterschiede zwischen Matchings möglich - Beim Merge muss entschieden werden, wie mit fehlenden Elementen umgegangen wird - Art gefundener Unterschiede sowie deren Vor- und Nachteile notieren

Use Case: UC-3	**Name:** Namen vergleichen
Kurzbeschreibung:	Vergleich von Namen/Bezeichnern zum Identifizieren ähnlicher Elemente, um ein grundlegendes Matching der beiden Ausgangsdiagramme zu erreichen, und Identifizieren semantischer Unterschiede. Funktionalität, Aufgabe und Zweck ähnlicher Elemente werden verglichen und semantische Unterschiede herausgearbeitet.
Anwendungsbereich:	Analyse der Ausgangsdiagramme
Ebene:	Nutzer
Primärer Akteur:	Proband
Projektbetroffene u. Interessen:	Proband will grundlegendes Matching der Ausgangsdiagramme identifizieren. Semantische Unterschiede werden implizit durch Namensähnlichkeiten offengelegt.
Vorbedingungen:	Ausgangsdiagramme liegen vor.
Erfolgsgarantien:	Matching von Elementen mit Namensähnlichkeiten identifiziert. Semantische Unterschiede bei Namensähnlichkeiten herausgearbeitet. Gibt es innerhalb dieser Ähnlichkeiten Namensunterschiede, so werden diese identifiziert.
Auslöser:	Proband vergleicht Ausgangsdiagramme.
Hauptszenario:	1. Proband vergleicht Namen in den Ausgangsdiagrammen. 2. Proband identifiziert Matching von zwei Elementen aufgrund von Namensähnlichkeiten. 3. Proband erarbeitet Namensunterschiede des Matchings und notiert Vor- und Nachteile. 4. Proband erarbeitet semantische Unterschiede des Matchings und notiert Vor- und Nachteile.
Erweiterungen/Alternativen:	2a. Kein Matching gefunden. 2a1. Element ist ein Kandidat für ein fehlendes Element. 2a2. Proband verifiziert fehlendes Element. 2a3. Abbruch des Hauptszenarios. 2b. Ähnlichkeit von 100% festgestellt. 2b1. Identische Elemente identifiziert, kein Namensunterschied. Weiter bei 4.
Dazugehörige Informationen:	Kategorie 1.1.1 Identifizieren von Namensähnlichkeiten, 1.2.3 Identifizieren von sem. Unterschieden
Offene Punkte:	Wie behandelt man den Parameter 'Granularität'? Vergleich auf Konstrukt-/Struktur-, Klassennamen- oder Parameterebene.
Priorität:	Hoch (einfachster, naheliegendster und am häufigsten verwendeter Vergleich)
Sonstiges, Anmerkungen:	- Ähnlichkeiten auf grober Ebene können in späterem Schritt um feinere ergänzt werden - In Kombination mit anderen Vergleichstechniken (Semantikvergleich) - Namensvergleich trotz unterschiedlicher Funktionalität möglich

Use Case: UC-20	**Name:** Matching visualisieren
Kurzbeschreibung:	Strategien zum Aufzeichnen/Festhalten von Beobachtungen bezüglich der Ähnlichkeit zweier Elemente. Markieren zweier Elemente gleichen Typs (auch Gruppen von Elementen). Markieren verteilter Funktionalitäten (Gruppen). Markieren expliziter/impliziter Realisierungen (semantisches Matching).

Anwendungsbereich:	Markieren der Ausgangsdiagramme
Ebene:	Nutzer
Primärer Akteur:	Proband
Projektbetroffene u. Interessen:	Proband möchte Matchinginformation graphisch festhalten, um möglichst ohne Informationsverlust mit weiteren Vergleichen fortzufahren und auf seine Visualisierung des Matchings zurückgreifen zu können.
Vorbedingungen:	Matching zwischen Elementen identifiziert
Erfolgsgarantien:	Identifiziertes Matching visualisiert
Auslöser:	Proband möchte Matchinginformation graphisch festhalten (visualisieren).
Hauptszenario:	1. Proband markiert in beiden Diagrammen Elemente oder Elementgruppen gleichen Typs nach gegebener Markierungsstrategie.
Erweiterungen/Alternativen:	1a. Markierungsstrategie ist abhängig von der Art der Ähnlichkeit: - Markieren ähnlicher Elemente - Verteilte Funktionalitäten - Explizite/implizite Realisierung
Dazugehörige Informationen:	Kategorie 2.3 Visualisierung eines Matchings, 2.3.1 Markieren ähnl. Elemente gleichen Typs, 2.3.2 Markieren verteilter Funktionalitäten, 2.3.3 Markieren expl./impl. Realisierungen
Priorität:	Mittel (Probanden, die diese Strategie nicht verfolgt haben, mussten sich in der Vereinigungsphase teilweise erneut mit den zuvor betrachteten Problemen auseinandersetzen)
Anm. zu physik. Artefakten:	- Mehrere der zur Verfügung gestellten Farben teils in Kombination mit unterschiedlichen Linienarten oder Formen verwendet: 1. Gefundene Matchings wurden in jedem der Ausgangsdiagramme mit der gleichen Farbe markiert. 2. Farben beschreiben nicht das Matching selbst, sondern dessen Art/Typ. Beispielsweise wurden alle Matchings bezüglich ähnlicher Elemente gleichen Typs mit der gleichen Farbe markiert. - Verweise auf ein Element im anderen Diagramm - Markieren von Gruppen mittels Großbuchstaben
Sonstiges, Anmerkungen:	- Unterschiedliche Markierungsstrategien - Visualisierungen lassen sich in einem späteren Schritt dazu verwenden, eine detailliertere Suche von Ähnlichkeiten durchzuführen

Use Case: UC-39	**Name:** Mergebegründungen auswählen
Kurzbeschreibung:	(Vereinigungs-)Begründungen, die Probanden während des Gesamtprozesses entwickeln. Im Allgemeinen beruhen Begründungen auf der Bewertung des Designs der Ausgangsdiagramme. Es können unterschiedliche Entscheidungen miteinander kombiniert werden oder sich gegenseitig beeinflussen. Es können unterschiedliche Gründe für das selbe Ergebnis existieren.
Anwendungsbereich:	Gesamtprozess
Ebene:	Nutzer
Primärer Akteur:	Proband
Projektbetroffene u. Interessen:	Jede Vereinigungsentscheidung von Matchings beruht auf Begründungen, die der Proband zuvor entwickelt hat.
Vorbedingung:	Auf Basis zuvor identifizierter Matchings
Erfolgsgarantien:	Begründung zur Vereinigungsentscheidung eines bestimmten Matchings entwickelt
Auslöser:	Identifiziertes Matching soll vereinigt werden.
Hauptszenario:	1. Proband entwickelt eine Begründung, um ein identifiziertes Matching zu vereinigen. 2. Proband wählt eine der folgenden Begründungsarten oder eine Kombination aus mehreren: - Einfachheit/Übersichtlichkeit (Strukturelle oder semantische Eigenschaften) - Funktionalität erhalten (zusätzliche Funktionalität, Differenzieren von Funktionalität, Flexibilität) - Äußere Einflüsse berücksichtigen (Bezug zur Realität, Bezug auf Implementierung) - Vorherige Entscheidungen berücksichtigen - Schlechte Modellierung vermeiden - Unkritische Unterschiede berücksichtigen

	- Elemente ausschließen (Überflüssige Elemente, unklare Bedeutung) - Terminologie erhalten
Erweiterungen/Alternativen:	2a. Proband kann keinen der aufgeführten Gründe wählen. 2a1. Proband hat andere Gründe. Proband führt seine Gründe auf (Sonstige).
Dazugehörige Informationen:	Kategorie 5 Begründungen, 5.1 Einfachheit/Übersichtlichkeit, 5.1.1 Strukturelle Eigenschaften, 5.1.2 Semantische Eigenschaften, 5.2 Funktionalität, 5.2.1 Zusätzliche Funktionalität, 5.2.2 Differenzieren von Funktionalität, 5.2.3 Flexibilität, 5.3 Äußere Einflüsse, 5.3.1 Bezug zur Realität, 5.3.2 Bezug auf Implementierung, 5.4 Beeinflussung durch vorh. Entscheidungen, 5.5 Schlechte Modellierung, 5.6 Unkritische Unterschiede, 5.7 Ausschließen von Elementen, 5.7.1 Überflüssige Elemente, 5.7.2 Unklare Bedeutung, 5.8 Terminologie
Offene Punkte:	- Anwendungsbereich zu vage
Priorität:	Mittel (Begründungen werden allerdings nicht immer explizit genannt)
Sonstiges, Anmerkungen:	- Werden hauptsächlich in Verbindung mit den Vereinigungsentscheidungen wie dem Auswählen, Hinzunehmen oder Verwerfen von Elementen verwendet - Spielen eine Rolle bei der Entwicklung bzw. dem Einbringen eigener Ideen oder falls Änderungen an bereits bestehenden Teilen des vereinigten Diagramms vorgenommen werden müssen - Gründe werden direkt vor der jeweiligen Vereinigungsentscheidung wiederholt - Vorbedingung für Mergeentscheidung

D Use Cases (in German) — Third Iteration (Detail Completion)

Use Case: UC-54	**Name:** Ausgangsdiagramme laden
Kurzbeschreibung:	Benutzer gibt Dateipfade zu beiden Ausgangsdiagrammen in bekanntem Format an. MMT liest die Ausgangsdiagramme ein und stellt sie geeignet im Workspace dar.
Anwendungsbereich:	Initialisierung
Ebene:	Nutzer
Primärer Akteur:	Benutzer
Projektbetroffene u. Interessen:	Benutzer lädt Ausgangsdiagramme in MMT, um mit dem eigentlichen Problem - dem Modelmerging - zu beginnen.
Vorbedingungen:	MMT wurde gestartet und Ausgangsdiagramme können geladen werden.
Erfolgsgarantien:	Ausgangsdiagramme wurden geladen und werden im Workspace geeignet angezeigt.
Auslöser:	Benutzer signalisiert MMT, dass er die Ausgangsdiagramme laden möchte.
Hauptszenario:	1. Benutzer gibt Dateipfad zu erstem Ausgangsdiagramm an. 2. Benutzer gibt Dateipfad zu zweitem Ausgangsdiagramm an. 3. Benutzer signalisiert MMT, dass beide Pfade angegeben wurden. 4. MMT liest Ausgangsdiagramme ein. 5. MMT stellt Ausgangsdiagramme geeignet dar.
Erweiterungen/Alternativen:	1a/2a. Dateiformat inkorrekt. 1a1./2a1. MMT signalisiert Benutzer, dass das Dateiformat inkorrekt ist und gibt auch Auskunft darüber, welches Dateiformat erwartet wird. Zurück zu 1./2. 3a. Benutzer hat nur einen Pfad zu einem Ausgangsdiagramm angegeben. 3a1. MMT signalisiert dem Benutzer, dass die Pfadangabe zum zweiten Diagramm fehlt. Weiter bei 1. bzw. 2.
Priorität:	Hoch

Use Case: UC-20	**Name:** Matching visualisieren
Kurzbeschreibung:	Benutzer möchte Beobachtungen bezüglich der Ähnlichkeit zweier Elemente graphisch festhalten. Dazu gehört das Markieren zweier Elemente gleichen Typs (auch Gruppen von Elementen), Markieren verteilter Funktionalitäten (Gruppen) und Markieren expliziter/impliziter Realisierungen (semantisches Matching). Vom Benutzer selektierte Matchings können vom MMT visualisiert werden, das MMT kann automatische Visualisierungen vornehmen oder es kann Matchings selektieren und dem Benutzer die Möglichkeit geben, dieses Matching dann auch zu visualisieren oder zu verwerfen.

Anwendungsbereich:	Markieren der Ausgangsdiagramme
Ebene:	Nutzer
Primärer Akteur:	MMT
Projektbetroffene u. Interessen:	Benutzer, MMT MMT unterstützt den Benutzer dabei, Matchinginformationen graphisch festzuhalten, um möglichst ohne Informationsverlust mit weiteren Vergleichen fortzufahren und auf die Visualisierung des Matchings zurückgreifen zu können.
Vorbedingungen:	In beiden Ausgangsdiagrammen sind jeweils Elemente selektiert bzw. vom MMT als Matching identifiziert.
Erfolgsgarantien:	Selektierte bzw. identifizierte Matchings sind visualisiert.
Auslöser:	Vorhandene Matchingsinformation wird graphisch festgehalten (visualisiert)
Hauptszenario:	1. Benutzer weist MMT an, sein selektiertes bzw. das vom MMT selektierte Matching zu visualisieren. 2. MMT visualisiert automatisch auf Basis der Art der Ähnlichkeit (Markieren ähnlicher Funktionalitäten, Verteilte Funktionalitäten, Explizite/Implizite Realisierungen)
Erweiterungen/Alternativen:	1a. MMT befindet sich im *Auto*-Modus. 1a1. Weiter bei 2.
Dazugehörige Informationen:	Kategorie 2.3 Visualisierung eines Matchings, 2.3.1 Markieren ähnl. Elemente gleichen Typs, 2.3.2 Markieren verteilter Funktionalitäten, 2.3.3 Markieren expl./impl. Realisierungen
Offene Punkte:	Bessere Bezeichnung für den Auto-Modus
Priorität:	Hoch (Probanden, die diese Strategie nicht verfolgt haben, mussten sich in der Vereinigungsphase teilweise erneut mit den zuvor betrachteten Problemen auseinandersetzen -> Vermeiden von Redundanzen in der Software. Weiterhin sind Visualisierungen für spätere Schritte nötig)
Anm. zu physik. Artefakten:	- Mehrere der zur Verfügung gestellten Farben teils in Kombination mit unterschiedlichen Linienarten oder Formen verwendet: 1. Gefundene Matchings wurden in jedem der Ausgangsdiagramme mit der gleichen Farbe markiert 2. Farben beschreiben nicht das Matching selbst, sondern dessen Art/Typ. Beispielsweise wurden alle Matchings bezüglich ähnlicher Elemente gleichen Typs mit der gleichen Farbe markiert - Verweise auf ein Element im anderen Diagramm - Markieren von Gruppen mittels Großbuchstaben
Sonstiges, Anmerkungen:	- Ausgangspunkt ist, dass jede Matchinginformation visualisiert werden sollte - Art der Ähnlichkeit lässt sich somit automatisch auf 1:1, 1:n und n:m Matchings übertragen. Somit lassen sich auf dieser Grundlage die unterschiedlichen Markierungsstrategien auswählen und MMT kann automatisch die Visualisierung vornehmen

Use Case: UC-1	**Name:** Ausgangsdiagramme vergleichen
Kurzbeschreibung:	Vergleich der Ausgangsdiagramme. Identifikation/Analyse/Beurteilung der Ähnlichkeit und Unterschiede beider Ausgangsdiagramme und der darin enthaltenen Elemente. Ziel ist es häufig, eine Grundmenge von Matchings zu identifizieren. Existiert ein Element nur in einem der Diagramme, so wird dieses als fehlendes Element identifiziert. Kann kein Matching mehr gefunden werden, so ergibt sich aus den übrigen Elementen der Unterschied.
Anwendungsbereich:	Analyse der Ausgangsdiagramme
Ebene:	Zusammenfassung
Primärer Akteur:	Benutzer
Projektbetroffene u. Interessen:	Benutzer möchte Matchings und Unterschiede mit Hilfe des MMT finden und visualisieren. Dies ist eine Voraussetzung für den späteren Merge der Ausgangsdiagramme.
Vorbedingungen:	Ausgangsdiagramme wurden in den Workspace geladen.
Erfolgsgarantien:	Matchings, Unterschiede und fehlende Elemente im Datenmodell gespeichert. Visualisierungsprozess durchlaufen.
Auslöser:	Benutzer startet Vergleich der Ausgangsdiagramme.
Hauptszenario:	1. Benutzer wählt Vergleichsart aus - Namen vergleichen

	- Layout vergleichen - Semantik vergleichen - Struktur vergleichen 2. MMT führt Vergleich durch und speichert Matchings im Datenmodell (fehlende Elemente und Unterschiede implizit). 3. MMT selektiert identifizierte Matchings sequenziell. 4. MMT visualisiert Matching. 5. Weiter mit nächstem Matching in 3.
Erweiterungen/Alternativen:	1a. Benutzer selektiert manuell ein Matching. 1a1. MMT führt keinen automatischer Vergleich aus. Weiter bei 4. 3a. MMT ist im Auto-Modus. 3a1. Keine Selektion notwendig. Weiter bei 4. 5a. Lokale Unterschiede sollen visualisiert werden. (*Modus?*) 5a1. MMT visualisiert lokale Unterschiede.
Dazugehörige Informationen:	Kategorie 1 Vergleich, 1.1 Matchings identifizieren, 1.2 Unterschiede identifizieren, 1.2.1 Identifizieren fehlender Elemente
Offene Punkte:	- Modus für lokale Unterschiede?
Priorität:	Hoch (essentieller Bestandteil des Gesamtprozesses; wichtiger Prozess)
Sonstiges, Anmerkungen:	- Ablauf von global nach lokal (Granularität von grob nach fein) - Unterschiede werden während des Merges näher untersucht - Unterschiede zwischen Matchings möglich - Beim Merge muss entschieden werden, wie mit fehlenden Elementen umgegangen wird - Art gefundener Unterschiede sowie deren Vor- und Nachteile notieren (siehe lokale Unterschiede visualisieren) - Grundlegende Informationen über Diagramme und die Domäne bereits bekannt -> Unterstützend - Navigation innerhalb der Diagramme notwendig -> Unterstützend (auch für den Algorithmus) - Granularität wird in den jeweiligen Vergleichs-Use Cases behandelt. Je nach Vergleichsart z.B. Namen: Untere und obere Grenze der Betrachtungsebene, Auswahl von bestimmten Elementtypen etc.; z.B. Layout: Klassenintern oder global - Entwicklung eines weiteren Use Cases `UC-55 Fehlende Elemente visualisieren` (inkl. Notiz der Vor- und Nachteile) - Globaler Unterschied = fehlende Elemente