Preprint. The final version of this paper will appear in the Proc. of ESEM 2011

How Humans merge UML-Models

Rainer Lutz, David Würfel and Stephan Diehl
Department of Computer Science
University of Trier
Trier, Germany
lutzr@uni-trier.de, s4dawuer@uni-trier.de, diehl@uni-trier.de

***Abstract*—So far, research on model merging has mostly focused on algorithmic problems. But, there are various situations when software engineers have to compare and merge different models manually or at least make important decisions. In this paper, we provide insights into the process of how users compare and merge visual models. To this end, we observed people's activities when manually merging UML class diagrams and analyzed the recorded data following the Grounded Theory approach. To illustrate the usefulness of our results we derived some guidelines for tool design.**

***Keywords*-merging; model; UML; grounded theory;**

I. Introduction

There are various situations when software engineers have to compare and merge different models. For example, several teams may have independently modeled the same system or different aspects or parts thereof, or new and old versions of the same model exist, or models have been automatically or manually reverse engineered from source code. Model comparison and merging may also be used to identify common architectural principles in the models of different systems or help to integrate those systems. In this paper we look at models in form of UML class diagrams, but many of our findings may also apply to other graph-based, visual models in software engineering.

In recent years, tools have been developed for configuration management to automatically or semi-automatically merge different versions of the same model [1], [2]. Also, some UML tools allow to load several models into the same view and then manually merge them. In the words of Barrett et al. [3]: "Put bluntly, the state of model merge tools is abysmal." But, how should such tools look like? What kind of information, what visualizations, what functionalities, what interactions should they provide? These questions finally lead to the more fundamental research question: *"How do software engineers merge UML models?"*

In this paper, we provide some insights into the process of how users compare and merge visual models. Finally, our results should enable more informed design of interactive tools for model merging and serve as a starting point for further empirical studies. To this end, we observed people's activities when manually comparing and merging UML class diagrams using pens and paper-based diagrams. This allowed us "to view their process independently of the confounds of a specific" tool [4]. To systematically analyze the recorded data we followed the Grounded Theory [5] approach, an established qualitative research method in social sciences.

The contributions of our work are a qualitative theory of human model merging activities and derived guidelines for tool design.

II. Experimental Design

In our experiment the participants had to merge two given UML class diagrams. While this task followed straightforward from our research questions, it was less obvious what diagrams to use for the experiment.

A. UML Class Diagrams

In a group discussion guided by the different UML model elements we came up with a catalog of alternative ways to model the same or similar aspects. This catalog includes the following alternative representations:

- same semantics, different names
- same names, different semantics
- different level of abstraction
- same element is part of different classes
- abstract class instead of interface
- use of explicit collection classes
- association instead of operation/aggregation/classes
- direction of associations
- primitive type instead of separate class
- inheritance in contrast to delegation

Next, we browsed through a pile of UML diagrams produced by students of our software engineering course. As part of their homework exercise they modeled an automatic parking barrier. From these diagrams we selected 2 pairs of matchable diagrams which covered many of the alternatives of the catalog. As a side effect, we also modified and extended the catalog. Moreover, we designed two UML models of a restaurant based on an example in a UML textbook to cover additional alternatives of the catalog.

B. Procedure

In total we recruited 13 participants for our study: one professional software developer, two undergraduates, six graduates and four PhD students. All participants were male at an average age of 29 years. According to their own

assessment in a post-study questionnaire, they have good to advanced UML experience and advanced experience in object-oriented programming.

With 7 participants we performed single-person Thinking Aloud sessions [6], [7] following the speech-communication approach of Boren and Ramey [8] using probing questions to remind the participant to verbalize their thoughts. With the remaining 6 participants we performed two-person sessions following the constructive interaction (aka. co-discorvery) approach [9].

The participants/groups were asked to merge both two parking-barrier UML class diagrams randomly selected from the two pairs mentioned above and the two restaurant UML models. There were no explicit questions to be answered and the test persons were allowed to ask clarifying questions. Thus, all participants were free to develop and follow their own strategies.

Figure 1. Annotation of UML class diagrams

As shown in Figure 1, the participants got a short requirements specification, a set of color pens, and three A3 sheets of paper: two with the printed diagrams and one to draw their merged model on. The sessions were recorded (video and audio). In addition, we were prepared to make some notes of unexpected events during the sessions. Finally, after completing both tasks, the participants had to fill in a questionnaire to collect general information about the test persons and more qualitative data relating to the research question.

C. Data Analysis

After the experiment, we had videos, merged models, and annotated original diagrams, as well as the questionnaire data. To systematically analyze the video data, we followed the Grounded Theory [5] methodology (*GT*):

1) Transcribe videos: identify time intervals, describe activities within each time interval;
2) Identify concepts (naming of phenomena/activities), annotate time intervals with concepts, and group concepts into categories (*GT: open coding*);
3) Identify further categories and their subcategories (classification); determine relations between categories; describe each category and define its properties (*GT: axial coding*);
4) Identify core category and arrange other categories around core category to form a "theory" (*GT: selective coding*).

Based on this theory we derived guidelines for tool design.

III. Results

We conducted ten experiments and gathered more than 7.5 hours of video data with an average duration of about 47 minutes per video. Those recordings were reviewed by two of the authors independently. In order to capture all activities and comments of the test persons, both researchers transcribed all video recordings to text files, i.e., for each experiment there exist two independently created transcriptions. A single transcription is basically a table that lists time stamps along with the according description as shown for the excerpt in Table I. Moreover, we annotated prominent activities and comments with early interpretations and hypotheses. For the three co-discovery experiments we also included an identifier for each test person.

Table I shows a transcription of a four minute period of a single test person. During that period he investigated specific parts of both restaurant models, which are depicted in Figure 2 along with several marks to show similarities or differences. Please note that not all of them were drawn during the four minute time period. Next, we describe this excerpt in more detail.

In a previous more coarse grained analysis the test person marked the `RecipeBook` class with a red color to indicate that this element is missing in the first class diagram (R1). Now, when he performs more detailed investigations, he recognizes the `recipeBook` attribute of the `Cook` class in R1 and matches it with the according class of the second diagram (R2) by highlighting both purple (A). Shifting his focus of attention to the recipe book he defers the current problem, starts investigating and comparing how this feature is realized in both diagrams, and identifies distinct structural differences. Nevertheless, in a next step, he concentrates on marking similarities between both recipe book versions (B). This done, he returns to the `Cook` class, quickly investigates the `prepare()` method, and switches to the relationships between `Cook` and `Chef`. There, he figures out that a `Chef` is a specialization of a `Cook` in R1. In contrast, R2 does not provide such a relationship although the two individual classes share the same functionality (C). Further investigation of those classes uncovers that there is no corresponding element in R1 for the `addRecipe()` method—neither for the `Cook` nor the `Chef` class of R2. Therefore, he underlines

	Time	Action/Comments
A	00:29:09	Investigating `Cook` classes → discovering a `recipeBook` as attribute of `Cook` in R1 and changing the mark (from red to purple)
B	00:29:28	Investigating `Recipe`/`RecipeBook` classes → R2 contains the more sophisticated one → different realization (generic vs. specific lists) → obvious/distinct differences → marking of all important inner similarities (those that are important for the realization of a recipe book) with blue lines and circles → matching of the name of the association between `Cook` and `Recipe` with three `add...()` methods of the `RecipeBook` class and the association itself with the aggregations between `RecipeBook` and the three recipe classes (`Appetizer, MainDish, Dessert`)
C	00:30:53	Back in the `Cook` class → via `prepare()` method investigation of relationship between `Cook` and `Chef` → inheritance (R1) vs. two individual classes, but with same functionality (R2)
D	00:31:29	No explicit match for the `addRecipe()` method of classes `Cook` and `Chef` (R2) exists in R1 → underlining with red color
E	00:31:56	Comparison of supervisor feature → different realization → reflexive association in R1, separate `Supervisor` class in R2 → marking of this matching with four blue diagonal lines one for the `supervises` relationship (R1), the `Supervisor` class (R2), and both `Chef` classes
F	00:32:37	Advantage of R2 is that only a `Chef` is allowed to be a `Supervisor`, in R1 such special relationships are not defined → drawback of R2 is the redundancy of classes (`Cook` and `Chef`)

Table I
EXCERPT OF TRANSCRIPTION

this method with a red color (Figures 1 and 2) (D). Next, he compares the realization of the supervisor feature and finds out that it is modeled implicitly with a reflexive association in R1 while R2 uses an explicit `Supervisor` class. Again, he concentrates on visualizing similarities (E). As a last step, he tries to identify advantages and disadvantages of each realization (F) and finishes his investigations concerning this part of the diagrams.

Transcriptions like the excerpt shown in Table I are not as detailed as possible, i.e., they do not always cover the exact wording of a test person (or each specific activity). But, our goal was not to transcribe the recordings to text files and afterwards put the videos away as it might be possible with voice recordings. Instead, we believe that for a detailed analysis you have to use both the transcriptions to navigate through a large amount of data and pick out a specific phenomenon, and the video recordings itself to further investigate that phenomenon. Especially interactions between a test person's comments and the performed activities can be analyzed in more detail by watching the video and, moreover, are hard to capture in a simple transcription.

For further reduction of the data, we analyzed our transcriptions line-by-line and annotated them with concepts in order to name the phenomenon/activity (*GT: line-by-line coding*). In particular, we walked through the text files, watched the according videos, and labeled the discovered incidents with a concept. Before we did the last step, we compared each incident with the existing concepts (*GT: constant comparison*) to determine whether it already occurred during the analysis or a new concept is needed.

For the example above we identified the concepts shown in Table II. Due to space limitations, we only take a closer look at three of these concepts.

Con1: A test person identified similar features in both models that were implemented implicitly in the first diagram and explicitly in the second one or vice versa. In this context the term *explicit* implies that additional classes were used to realize a certain feature, while *implicit* means that this

ID	Name of Concept	Timestamps
Con1	Identification of explicit/implicit realizations	A,E
Con2	Identification of semantic differences	B
Con3	Identification of structural differences	B,C
Con4	Identification of distributed features	B
Con5	Marking of explicit and implicit realizations	A,E
Con6	Marking of distributed features	B
Con7	Marking of missing methods	D
Con8	Switching to related problem	B
Con9	Switching to original problem	C
Con10	Reconsideration of previous decisions	A
Con12	Uncovering weaknesses	C
Con13	Studying of advantages and disadvantages	F

Table II
CONCEPTS IN THE EXAMPLE.

can be achieved without them. For instance, in Figure 2 a specific `RecipeBook` class was modeled in R2, while a simple attribute of the type `Set` is used in R1.

Con8: This concept describes the activity when test persons moved to a different, but related problem and deferred the current one. In most cases they returned to the original task after the newly discovered problem was solved (Con9).

Con12: Especially when comparing the models, test persons sometimes uncovered weaknesses concerning the design of a diagram. As a further step participants often studied and compared advantages and disadvantages in case they had already identified corresponding elements in the source diagrams (Con13).

The first iteration of our analysis exposed over 180 concepts, which were grouped into categories and subcategories during further steps. In addition, we derived properties to describe different characteristics of a category (*GT: open coding, axial coding*).

IV. FINDINGS

While we identified and developed the categories bottom-up, we describe them top-down for better comprehensibility. Before we present our results in detail, we want to outline how a typical merging of UML class diagrams is performed.

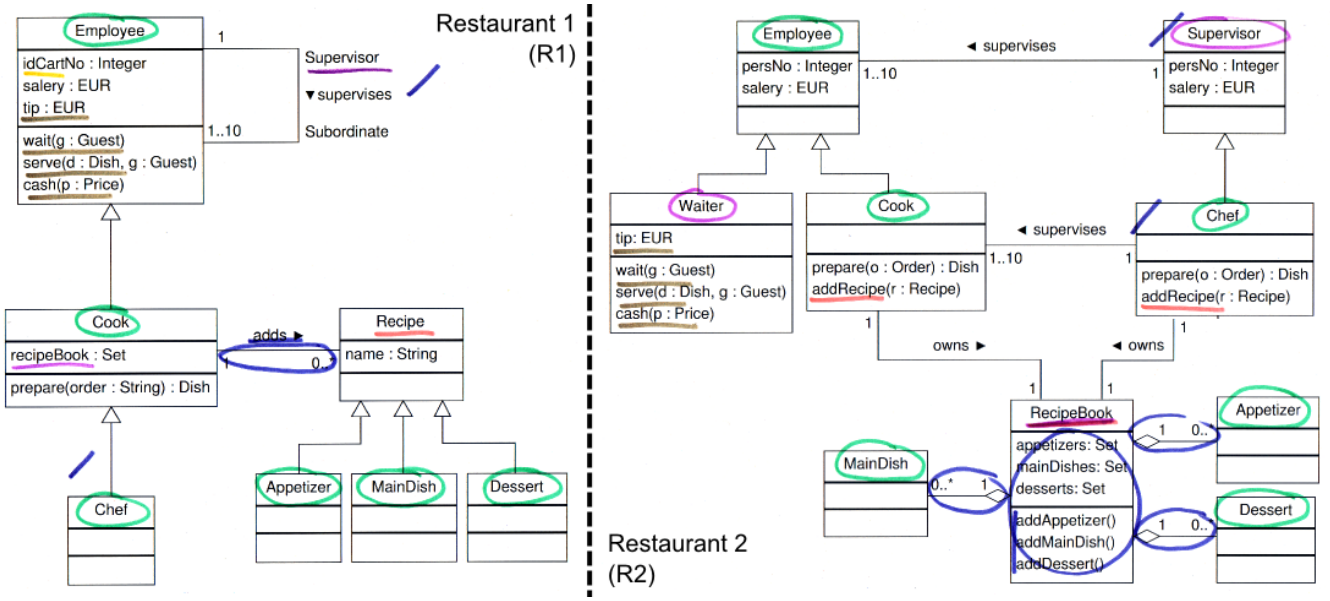

Figure 2. Excerpts of annotated restaurant UML class diagrams. Here, different colors and shapes indicate different types of similarities and differences.

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 visualized 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 test persons developed rationales that helped combining the source diagrams. In addition, new/own ideas might be considered to improve the merged diagram. Finally, test persons checked whether all elements have been investigated and some even validated the newly created UML class diagram.

When merging UML class diagrams, the most important step is to figure out what to merge—to ask questions like "Are there any obvious similarities or differences about the models?" or "How is this specific feature realized in the other diagram?". In other words, the goal is to compare the models, identify corresponding elements (matchings), and use this information to finally merge the source diagrams. Therefore, participants ran through two phases. During the *comparison phase* the source diagrams were compared and matchings were identified. In contrast, the *merging phase* covers activities in order to develop a merged diagram. Comparing is the essential process of model merging and, therefore, evolved as core category during our analysis. To record information gathered during comparison some test persons developed different kinds of annotation and visualization techniques. Next, we present a detailed description of the *Comparison* category and, furthermore, discuss its relationship to the *Visualization* category.

A. The Core Category

Cat1: Comparison. During the comparison phase the source diagrams are examined for both similarities and differences. The gathered information is essential for subsequent steps like visualization or merging. Please note that the comparison phase is mainly a continuous process at the beginning of an analysis. Later, it may alternate with the merging phase because specific elements need to be studied and compared in detail. Generally, test persons tended to start with a global overview or a coarse grained comparison and proceeded until they reached a certain fine grained level, where they investigated local similarities (e.g., the content of classes). Coarse grained comparison covers classes and their relationships, while fine grained investigations consider attributes, methods, or even return types and edge labels. Depending on a test persons strategy, finer grained investigations were done directly or at a later time, for example, during the merging phase. Especially the first impression that arises from a global comparison can shape further strategies, for instance, where to begin with detailed investigations or which classes might possibly match.

Thinking about how to compare UML class diagrams and which strategies to follow one may assume that this problem can be reduced to a simple "Find 10 differences" game. In practice, diagrams to be merged often contain less differences than similarities. But especially at a coarse grained level, test persons concentrated on the identification of similarities rather than differences, which might be considered as an inconvenient strategy at first. And even

further, differences that were considered to be uncritical were sometimes completely ignored. Nevertheless, identifying differences is an essential step, particularly, when it comes to merging both diagrams.

Comparison includes two subcategories, which are introduced hereafter.

Cat1.1: Identification of Matchings: This subcategory comprises strategies for the analysis and assessment of the similarities between both source diagrams. The identification of matchings provides a basis for subsequent procedures including the identification of (local) differences, the visualization of matchings, and the development of a merged model. After basic information about the diagrams, their purpose, and the related domain were gathered, test persons started identifying corresponding elements, so called matchings. These matchings equate elements of the first diagram to elements of the second one. You may think of any reasonable assignment, e.g., not only two classes can be matched, but also a class with a group of classes, an attribute, or a relationship (Table I). In general, test persons tended to identify a basic set of mostly coarse grained (class-based) matchings—either consciously or intuitively—before they proceeded with a detailed analysis, fine grained investigations, or even the merging process. Especially at the beginning, a common strategy was to find coarse grained matchings and only perceive potential differences, which did not essentially contribute to such a matching decision.

This category can further be divided into four subcategories. The *Identification of Name* and *Layout Similarities* may be considered as simple strategies, which can be used in a fast and straightforward way, while the *Identification of Semantic* and *Structural Similarities* are more complex, therefore, require more effort but generally result in more detailed information. The order in which the strategies are presented here is not arbitrary, but roughly reflects the frequency of their usage in our experiments. Furthermore, these strategies were also applied in a combined manner. For instance, test persons who could not make a decision based on name similarities only, tried to identify semantic and/or structural similarities additionally.

Cat1.1.1: Identification of Name Similarities: The simplest way to match elements is to compare their names/identifiers. Here, some participants found it sufficient for two names to have only substrings in common or even looked for synonyms, while others preferred identical strings. However, this strategy was most frequently used and often chosen as the initial one. Especially when classes and their relationships were examined (class level), it was combined with other strategies to acquire a set of coarse grained matchings. Nevertheless, some test persons also matched elements with similar name but with different functionality.

Cat1.1.2: Identification of Layout Similarities: Mostly used in early stages of the comparison phase to support a global overview. The main purpose of layout similarities is to identify similar elements based on their relative positions. This may also work for local, class internal investigations where positioning mainly reflects the order of attributes, methods, etc.. In general, identification of layout similarities was combined with other strategies particularly name or semantic based ones and was rarely explicitly mentioned during our experiments. Nevertheless, this strategy might provide a crucial criterion for a matching.

Cat1.1.3: Identification of Semantic Similarities: The most frequently used complex strategy is the identification of semantic similarities. The exact definition of semantic similarities might be different for each test person, but, in general, elements were compared with respect to their functionalities, their responsibilities, or their purpose. Elements that share such semantic similarities were defined as matching—mostly independent from their actual realization. For instance, confer Timestamp B in Table I where both recipe book variants were matched based on their purpose. This shows that matched elements may differ to some extent. However, they were often reinvestigated in a further analysis.

Cat1.1.4: Identification of Structural Similarities: To identify structural similarities, participants had to investigate the relationships of the considered elements from a global point of view. Although structural similarities were rarely identified explicitly, they were often used to support the other strategies, especially semantic similarities. On the other hand, at least at the beginning of their analysis, some test persons tended to ignore structural properties. As with name similarities, elements that were considered to be similar with respect to structural properties might differ in functionality.

Next, we discuss the relation to the *Visualization* category. *Identification of Matchings* only comprises strategies to identify matchings and does not cover any visualization aspects. However, some test persons decided to annotate elements of the source diagrams with information gathered during their analysis in order to keep this information for later use.

In general, matchings were visualized based on the test person's individual approach directly after they had been identified. Most participants used different colors, shapes, and/or line types to visualize matchings. Especially colors were used in two ways: different colors either indicated different matchings or different types of matchings. For the latter, test persons 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 as described in Section III.

Cat1.2: Identification of Differences: This subcategory includes strategies for the analysis of differences between both source diagrams and the included elements. Differences were often discovered implicitly during the identification of matchings, i.e., they are mostly based on a local comparison of a specific matching. From a global point of view, test

persons tended to search for matchings until no further one can be identified, the rest was defined to be the difference. Although matchings seem to be of higher priority the identification of differences is essential for the comparison of UML class diagrams.

As with similarities, we divided *Identification of Differences* into several subcategories, which again are introduced ordered by the frequency of their usage, beginning with the most frequently used one.

Cat1.2.1: Identification of Missing Elements: An element is called missing if it exists in only one source diagram, i.e., no matching element could be identified in the other model. Usually, test persons did not aim to find missing elements, they remained after the identification of matchings. Later, during the merging phase participants had to decide if those elements shall be added to the merged diagram.

Cat1.2.2: Identification of Design Differences: Differences in design, modeling, or realization are generally based on previously identified semantic matchings. With respect to a later merging step, it is important to identify such differences and study related advantages and disadvantages. This subcategory covers among others concepts like Con1, Con3, and Con4 (Table II). As described at the end of Section III, we used properties to cover different characteristics of a category. Here, depending on a test person's strategy, elements with similar features were investigated with respect to their level of detail, distribution, and/or structure. *Level of detail* covers both elements of the same type, for example, a single method of the first diagram can be matched with several methods of the second one and different type like the identification of explicit/implicit realizations (Con1). *Distribution* describes situations when features are spread over several elements in the first diagram and agglomerated in a single element of the second one. Finally, *structure* means that elements can be connected to the rest of the diagram in different ways although they model the same features.

Cat1.2.3: Identification of Semantic Differences: Especially in the merging phase, semantic differences are often crucial to create the new diagram. Test persons had to decide whether a certain feature will be integrated or not. Again, this strategy is mainly based on previously identified matchings and, thus, considers local differences with respect to functionality or purpose. The bigger these differences, the more reasonable is a detailed analysis to gather advantages and disadvantages of a certain element or feature.

Cat1.2.4: Identification of Name Differences: Name differences were often identified along with matchings that are not based on name similarities. Especially during the merging phase, it may be beneficial to be aware of them, because participants faced the problem to assign a name to an element.

Cat1.2.5: Identification of Layout Differences: Like layout similarities, differences were mostly discovered during early stages of the comparison phase especially on a global level. Different layouts may complicate the comprehension and the comparison of the source diagrams [10] and, therefore, it is important to be aware of them. As with most differences, these were rarely identified explicitly, i.e., test persons discovered them while examining the models for similarities.

As with matchings, we briefly discuss the relation to the *Visualization* category because *Identification of Differences* does not cover any strategies for their visualization. Due to the fact that differences were rarely discovered explicitly, participants tended to mark similarities only and, thus, outline differences implicitly. If differences were visualized explicitly, it was mainly done on a local level between elements of previously identified matchings with the intention to save this information for further steps. But, considering the example above (Table I, Timestamp A), some test persons also marked missing classes on a global level.

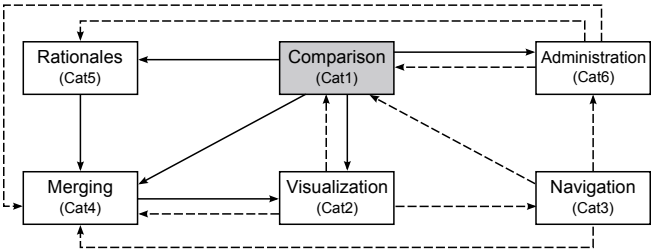

Figure 3. Relations between top level categories. Continuous arrows encode preconditions and dashed arrows depict supportive relations.

B. Integrating Categories

As described by Corbin and Strauss [11], for developing Grounded Theory the core category needs to be linked to all of the other categories that were identified during the previous coding steps (*GT: selective coding*). This subsection briefly introduces those categories and explains how they are related to the *Comparison* category and to each other.

Figure 3 depicts basic relations between the top level categories. There, we distinguish two types of edges. Continuous arrows show a precondition relation between two categories, i.e., a process of the source category has to be completed in order to proceed with strategies of the target category. Dashed arrows encode supportive relations, which might not be essential for the target category, but may facilitate a certain process. Please note, that an edge is drawn when at least two subcategories are related to each other. In other words, an edge does not necessarily mean that all subcategories are linked.

Cat2: Visualization. Besides the visualization of matchings and differences this category covers additional strategies to visualize and record collected information. This information may support both a further, more detailed comparison and the merging process itself (cf. Fig. 3). In general, test persons tended to apply visualization strategies more often

when the complexity of the diagrams increased. Furthermore, visualization may also be applied on different levels of granularity. Next, we introduce the subcategories:

Cat2.1: Auxiliary Gestures were applied by almost all test persons. Their main purpose was to focus elements or keep track of a certain observation (like when reading a book and using a finger to keep track of the current line). Gestures were used in different situations, for instance, to support the navigation through a single diagram or the investigation of elements and possible matchings. Besides pointing gestures participants also applied a temporary grouping of elements through grabbing or circle gestures.

Cat2.2: Visualization of the Progress: A widely used technique that requires a permanent annotation of the source diagrams was the visualization of the current merging progress. Test persons tried to facilitate their navigation through the source diagrams by focusing on elements that had not been processed. To this end, they tended to simply strike out elements within the source diagrams that were covered in a previously completed merging step (cf. Fig. 3). Some participants also distinguished added and discarded elements.

Cat2.3: Grouping of Elements was done for different reasons, in a temporary or permanent way. For example, most test persons identified matchings based on distributed features (1:n matching). In order to visualize such a matching, they decided to group the respective elements and treated them as a single element. From a global point of view, grouping was mainly used to abstract from a certain level of detail, to simplify the actual problem, or to augment the source diagrams with additional, but helpful information (cf. Fig. 3). In a Divide-And-Conquer manner some test persons divided the source diagram into groups, which were analyzed separately at first and afterwards the relationships in between them.

Cat3: Navigation. Besides navigating through a single diagram, participants mostly faced the problem to traverse both diagrams at the same time. In other words, during the comparison and merging phases strategies for navigation had to be synchronized to some extent. For instance, while merging, participants had to navigate from previously considered elements to the next matching synchronously. Therefore, *Navigation* is essential for most processes of model merging and closely coupled with *Comparison* and *Merging* (cf. Fig. 3). In particular, this category comprises strategies that were mainly developed and used implicitly during the overall process and may also be combined.

Cat3.1: Selection of a Starting Point: Before test persons were able to analyze the source diagrams, they often faced the problem of how to determine a starting point. Especially at the beginning of the comparison and merging phases, initial elements or matchings had to be chosen in order to navigate through the diagram. Therefore, the properties of this subcategory reflect three different ways to determine a starting point. Elements were preferably chosen as such when they were positioned closer to the upper left corner, provided essential features, or showed obvious similarities or differences.

Cat3.2: Structure-Based Strategies describe the actual navigation within the source diagrams, i.e., how participants moved from one element to another. A structure-based navigation was applied on different levels of granularity and, thus, may be adapted, but is generally grounded on the underlying visual graph structure. In other words, test persons followed relationships between classes to determine the next element to consider. This was of course not always done as strictly as by a traversal algorithm, but, nevertheless, a depth first search strategy was often applied. Inside classes test persons mainly proceeded in a list-based manner. In case a synchronized navigation within both source diagrams was needed, test persons often resort to auxiliary pointing gestures.

Cat3.3: Supporting Strategies: In addition, some test persons also developed approaches to simplify the navigation in order to keep track of the actual problem. In particular, they often used some kind of group based navigation, where they moved from one group to another after they had partitioned the diagrams (Cat2.3, Fig. 3). Less common was the approach of moving along different levels of granularity. For example, participants started to investigate only classes, followed by their relationships, and, finally, their content.

Cat3.4: Interrupting Strategies describe situations when participants followed a certain navigation strategy, but for some reason paused this procedure, investigated a different part of a source diagram, and, finally, returned to the original problem. Test persons navigated to a different element due to the following reasons: (1) Difficulties while merging the diagrams could make participants switch to an element that was easier to handle. (2) Test persons were distracted by a more interesting or detailed element. (3) When participants had treated an element that did not have any more relationships to follow (leaf node), they often switched to the next closest element (in terms of distance in the layout).

Cat4: Merging. This category comprises only strategies that were applied during the merging phase in order to combine the source diagrams and, therefore, merging is the final step of the overall process and essential to yield the new model. *Merging* implies that a comparison of the corresponding elements has been already performed, matchings identified, and rationales for their further treatment derived. In addition, previously made visualizations may support the merging process. Please note that *Merging* is rarely a continuous process, instead test persons tended to constantly alternate between the comparison and merging phases.

Cat4.1: Merging Decisions: Based on the information provided by *Comparison* and *Rationales* (cf. Fig. 3), test persons were able to decide whether elements shall be integrated into the merged diagram or discarded. In general,

merging decisions were applied on any level of granularity and can be classified into one of the following subcategories: (1) choosing an element, (2) adding a missing element, or (3) discarding a missing element. When choosing an element, test persons resorted to previously identified matchings (e.g., `idCartNo` ↔ `persNo` in Figure 2) and derived rationales in order to decide, which element shall be added to the merged diagram. Missing elements are only part of one diagram and, depending on previously elaborated rationales, participants either added this element to the new model or discarded it. Discarded elements may be crossed out within the source diagrams to visualize the overall progress.

Cat4.2: Drawing the Merged Diagram: This subcategory covers the actual drawing of the merged diagram, which implies that all merging decisions had already been done in a previous step and test persons adhered to them. In general, we differentiate between local and global drawing strategies, which may also be used in combination. In the former case, elements were integrated into the merged diagram directly after the respective merging decision was made. When applying global strategies, participants draw several class boxes with its name only and might connect them with simple lines in order to achieve a basic layout or to keep track of the identified matchings. After elements were drawn the overall progress might be visualized (cf. Fig. 3).

Cat4.3: Introducing New/Own Ideas: Own ideas were introduced due to the following reasons and were applied on different levels of granularity: First, the merged diagram had to be adjusted in order to preserve a certain feature, semantics, and/or validity of the model. Second, test persons discovered features that are neither available in the source diagrams nor achievable through merging or, in contrast, appear in both models, but were considered unnecessary for any reason. Furthermore, we observed that participants of co-discovery experiments found it more common, often due to more intense discussions, to add or remodel features.

Cat4.4: Changing the Merged Diagram: Merging decisions could turn out to be erroneous, for instance, further investigations affected previous decisions. Therefore, it might be necessary to modify the merged diagram and discard parts of the current model.

Cat5: Rationales. By analyzing and assessing the design of the source diagrams based on previously identified matchings, test persons developed different rationales, which emerged as an important precondition for *Merging* (cf. Fig. 3). Especially when considering merging decisions, such rationales could occur at various points in time and, thus, might not be related directly to the final merging step. Moreover, some rationales were combined with or were influenced by each other. Due to subjective influences different rationales may exist for the same result.

Cat5.1: Simplicity: Test persons preferred the simplest realization as long as the demanded feature was included. In particular, they checked for or even developed a solution that reflects the requirements as good as possible without retaining additional or unnecessary features.

Cat5.2: Functionality: This subcategory covers cases when participants wanted to preserve (additional) features, which might not be a requirement at first. This may be done for different reasons, e.g., in order to retain a certain flexibility, which supports a further development or to differentiate features. Some test persons simply tried to integrate as many features as possible into the merged diagram.

Cat5.3: External Influences are rationales, which go beyond the information that is provided by the source diagrams. Test persons resorted to own experiences with the underlying domain, the reality and/or the implementation. External influences were stated the most and also affected other rationales or supported generating own ideas.

Cat5.4: Previous Decisions: Elements were also be chosen based on previous merging decisions. For instance, elements of the same diagram were usually easier to handle, because their relationship was already defined. Therefore, test persons tended to adopt such elements, relationships, or even whole parts of a diagram if they could not identify differences at a first glance or merging of both diagrams was hardly possible.

Cat5.5: Weak Design: From time to time, test persons discovered design weaknesses or even errors within the source diagrams. While this may be their subjective impression in the former case, errors are more critical. In order to create a correct solution, the respective elements were either discarded or, less common, improved by own ideas.

Cat5.6: Uncritical Differences: When test persons identified differences that were uncritical according to their own opinion, they often chose a simple, obvious, or generic solution. In general, the more fine grained their investigations, the higher the probability to define differences as uncritical.

Cat5.7: Unnecessary Elements: An element may be defined as unnecessary and, thus, was often discarded for two reasons: First, its functionality was clear, but it did not contribute important features, e.g., as a consequence of *Simplicity*. Second, participants were not able to work out the exact purpose of a specific feature.

Cat5.8: Terminology: Especially when features of matched elements were very similar (or even identical), test persons often chose elements based on their names/identifiers. To this end, they referred to criteria like comprehensibility or programming guidelines.

Cat6: Administration. During the overall process participants also performed what we call administrative tasks. Although they were not always essential for a successful model merging, they could facilitate it and aided to solve problems. Figure 3 shows that *Administration* supports *Comparison*, *Rationales*, and *Merging*.

Cat6.1: Information Gathering is an important process of each analysis to familiarize with the task and the provided materials. In general, information belong to one of the

following areas: (1) Basic information about the source diagrams themselves, their features and purpose, and the corresponding domain. Participants gathered such information mostly at the beginning of their analysis to get a first impression and a global overview. (2) Besides matchings and differences test persons also generated additional information while comparing the source diagrams. For instance, they studied advantages and disadvantages of a certain realization, which was important to find rationals (cf. Fig. 3). (3) Test persons tended to reflect about own ideas. Not all of these ideas were later integrated into the new diagram, but, nevertheless, this information supported the merging process. (4) Some participants planed and developed individual strategies explicitly, before they began with their analysis. Using this information during the overall merging process seemed to make them focus better on the actual task.

Cat6.2: Review and Validation: In order to merge the source diagrams correctly, participants resorted to review and validation strategies. Particularly at the end of the experiments, test persons tended to review the source diagrams to find forgotten elements. Less common was a validation of the merged model in terms of semantics and UML syntax, which could lead to changes of already integrated elements.

V. Guidelines

To demonstrate the usefulness of the theory described above, we derived initial guidelines for tool design.

Support individual workflow: While we tried to capture the essence of the merging process in our theory, there have been many individual differences in the way the participants compared and merged models. Thus, a tool for interactively merging UML class diagrams should not restrict users to a certain workflow. Furthermore, it should provide a variety of features users may choose from.

Allow extensions: Besides the actual merging all aimed to improve the merged model to some extent by introducing own ideas. Therefore, a tool should provide at least basic UML modeling features and not only focus on merging itself (Cat4.3).

Support annotations: Category 2 covers a variety of different visualization and annotation techniques, which should be integrated in some form into a tool. Furthermore, we observed that some participants did not want to incorporate complex ideas directly into the merged model. The possibility to add comments to both the source diagrams and the merged diagram could help the users to capture their ideas for later use (Cat6.1).

Support grouping: Interactive model merging tools should support grouping to match single elements with groups. Grouping can also be used to reduce the complexity of the source diagrams and to enable the users to work more efficiently [12]. (Cat2.3).

Raise awareness: Participants tended to work on a preferred source diagram and sometimes overlooked or ignored important differences. Therefore, a tool should point users to such differences in order to make them aware of possible problems without distracting their workflow. Furthermore, a tool may also indicate alternative matches to make the users reflect on their previous decisions (Cat5.4).

Provide algorithmic support: At least at the beginning of their analysis, test persons mainly identified matchings based on straightforward strategies like name or layout similarities. Thus, a primitive name based matching algorithm could already help to simplify the problem (Cat1.1).

Help to keep track: The test persons came up with various gestures and markings to keep track of where they were in the merging process. Thus, a tool should record the interaction history and allow the users to go back and forth. Furthermore, users should be able to interrupt operations and the tool should later on remind them of incomplete operations (Cat3.4).

VI. Related Work

Our extensive literature research did not reveal any work on how humans manually merge UML class diagrams, therefore, we review here related work in a broader context including model driven configuration management, layout of UML class diagrams, and our research method, the grounded theory.

Begel and Simon [13] conducted a study using grounded theory on professional novices in software development. In contrast to our study, they made a long term observation of their subjects in their normal work environment and were rather focusing on the amount of time subjects spent on doing typical software development tasks than on *what* subjects do.

Crabtree et al. [14] were using a grounded theory approach for gaining insights in how people verbalize software processes. They make elaborate use of the grounded theory by Strauss and Corbin [11] to present a well-founded map of codes in great detail for understanding of how participants proceed with respect to their research question.

Yusuf et al. [12] extended the eye-tracking study by Guehéneuc [15] and explored human comprehension of UML class diagrams. They found that "experts tend to navigate/explore from the center of the diagram to the edges whereas novices tend to navigate/explore from top-to-bottom and left-to-right". This phenomenon also appeared in our analysis.

Purchase et al. empirically investigated the layout and aesthetics of UML diagrams including user preferences [10] and preferable syntactic notations for better comprehension [16]. Sun and Wong [17] used Gestalt Theory to justify a large number of criteria for the layout of UML class diagrams.

The use of UML in general and its advantages in professional software engineering were surveyed by Nugroho and Chaudron [18]. Their respondents "believe that the correspondence between UML models and the implementation is

important" and "the use of UML is perceived to be most influential in improving the quality of software in terms of understandability and modularity respectively.". Furthermore, "amongst other factors, incompleteness in UML models is considered more often leading to implementation problems and more often driving deviations in the implementation", which allows us to state that it seems reasonable to individually create several UML class diagrams (of the same or coherent software artifacts) and merge them in a next step to afford a maximum degree of completeness.

VII. Threats to Validity

The diagrams used in our study are based on both student exercises and an adjusted example from a textbook. Therefore, they might not reflect UML class diagrams created by professional software engineers. Furthermore, not all design elements such as stereotypes or association classes were covered by our diagrams. With one exception, all of our test persons were students and not professional software architects. People, who have to merge models every day, may have developed additional strategies or abandoned inefficient ones.

VIII. Conclusion

In this paper we presented insights in how software engineers compare and merge visual models. Therefore, we designed two pairs of comparable class diagrams and conducted a combined thinking aloud/constructive interaction study. Following the grounded theory methodology, we analyzed the gathered data in order to find concepts and derive categories from those. These categories and their relations provide a map of various activities our test persons performed when trying to merge the source diagrams. In particular, we identified *Comparison* as the core category and grouped all other categories around it. In order to support future work on tool design, we derived initial guidelines based on our findings.

References

[1] U. Kelter, J. Wehren, and J. Niere, "A Generic Difference Algorithm for UML Models," in *Software Engineering 2005, Fachtagung des GI-Fachbereichs Softwaretechnik, 8.-11.3.2005 in Essen*. GI, 2005, pp. 105–116.

[2] P. Selonen, "A Review of UML Model Comparison Approaches," in *Proceedings of the 5th Nordic Workshop on Model Driven Engineering, 27-29 August 2007, Ronneby, Sweden*, 2007, pp. 37–51.

[3] S. Barrett, P. Chalin, and G. Butler, "Model merging falls short of software engineering needs," in *Proceedings of the Workshop on Model-Driven Software Evolution, Athens, Greece*, 2008.

[4] P. Isenberg, A. Tang, and M. S. T. Carpendale, "An exploratory study of visual information analysis," in *Proceedings of the 2008 Conference on Human Factors in Computing Systems, 2008, Florence, Italy, April 5-10, 2008*. ACM, 2008, pp. 1217–1226.

[5] B. G. Glaser and A. L. Strauss, *The discovery of Grounded Theroy: Strategies for Qualitative Research*. Aldine Transaction, 1967.

[6] K. Ericsson and H. Simon, "Verbal reports as data," *Psychological Review*, vol. 87, no. 3, pp. 215–251, 1980.

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

[8] M. Boren and J. Ramey, "Thinking aloud: Reconciling theory and practice," *IEEE Transactions on Professional Communication*, vol. 43, no. 3, pp. 261–278, 2000.

[9] N. Miyake, "Constructive interaction and the iterative process of understanding," *Cognitive Science*, vol. 10, no. 2, pp. 151–177, 1986.

[10] H. C. Purchase, J.-A. Allder, and D. A. Carrington, "Graph layout aesthetics in UML diagrams: User preferences," *J. Graph Algorithms Appl.*, vol. 6, no. 3, pp. 255–279, 2002.

[11] A. L. Strauss and J. Corbin, *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*.

[12] S. Yusuf, H. H. Kagdi, and J. I. Maletic, "Assessing the comprehension of UML class diagrams via eye tracking," in *15th International Conference on Program Comprehension, June 26-29, 2007, Banff, Alberta, Canada*. IEEE Computer Society, 2007, pp. 113–122.

[13] A. Begel and B. Simon, "Novice software developers, all over again," in *Proceeding of the Fourth international Workshop on Computing Education Research*, ser. ICER '08. ACM, 2008, pp. 3–14.

[14] C. A. Crabtree, C. B. Seaman, and A. F. Norcio, "Exploring language in software process elicitation: A grounded theory approach," in *Proceedings of the Third International Symposium on Empirical Software Engineering and Measurement, October 15-16, 2009, Lake Buena Vista, Florida, USA*. IEEE Computer Society, 2009, pp. 324–335.

[15] Y.-G. Guéhéneuc, "Taupe: towards understanding program comprehension," in *Proceedings of the 2006 conference of the Centre for Advanced Studies on Collaborative Research, October 16-19, 2006, Toronto, Ontario, Canada*. IBM, 2006, pp. 1–13.

[16] H. C. Purchase, L. Colpoys, M. McGill, D. A. Carrington, and C. Britton, "UML class diagram syntax: An empirical study of comprehension," in *Australasian Symposium on Information Visualisation, InVis.au,Sydney, Australia, 3-4 December 2001*. Australian Computer Society, 2001, pp. 113–120.

[17] D. Sun and K. Wong, "On evaluating the layout of UML class diagrams for program comprehension," in *Proceedings of the 13th International Workshop on Program Comprehension, 15-16 May 2005, St. Louis, MO, USA*. IEEE Computer Society, 2005, pp. 317–326.

[18] A. Nugroho and M. R. V. Chaudron, "A survey into the rigor of UML use and its perceived impact on quality and productivity," in *Proceedings of the Second International Symposium on Empirical Software Engineering and Measurement, October 9-10, 2008, Kaiserslautern, Germany*. ACM, 2008, pp. 90–99.