Comparing State- and Operation-based Change Tracking on Models

Maximilian Koegel*, Markus Herrmannsdoerfer†, Yang Li‡, Jonas Helming§ and Joern David¶
Institut für Informatik, Technische Universität München
Boltzmannstrasse 3, 87548 Garching, Germany
*Email: *koegel@in.tum.de, †herrmama@in.tum.de, ‡liya@in.tum.de, §helming@in.tum.de and ¶david@in.tum.de*

***Abstract*—In recent years, models are increasingly used throughout the entire lifecycle in software development projects. In effect, the need for collaborating on these models emerged, requiring change tracking and versioning. However, many researchers have shown that existing methods and tools for Version Control (VC) do not work well on graph-like models, such as UML, SysML or domain-specific modeling languages. To alleviate this, alternative techniques and methods have been proposed which can be classified into state-based and operation-based approaches. Existing research shows advantages of operation-based over state-based approaches in selected use cases, such as conflict detection or merging. However, there are only few results available on the advantages of operation-based approaches in the most common use case of a VC system: review and understand change. In this paper, we present and discuss both approaches and their use cases. Moreover, we present the results of an empirical study to compare a state-based with an operation-based approach for the use case of reviewing and understanding change. For this study, we have mined an operation-based model repository and interviewed users to assess their understanding of randomly selected changes. Our results indicate that users better understand complex changes in the operation-based representation.**

I. Introduction

Today, models are an essential artifact throughout the entire lifecycle in software engineering projects. Model-driven development is putting even more emphasis on models, since they are not only an abstraction of the system under development, but the system is (partly) generated from its models. Consequently, models are about to cover the whole development process from requirements over design to deployment—including management of the process itself. There is a variety of models in use, such as UML, SysML or even domain-specific modeling languages. With the adoption of model-driven development in industry, the need for managing these models in terms of change tracking and versioning emerged. Version Control (VC)—also commonly known as Software Configuration Management—is already in wide-spread use for textual artifacts such as source code.

However, many publications, e.g. [1], [2], [3], [4], [5], [6], [7], recognized that existing VC approaches do not work well on models, which are essentially attributed graphs. The traditional VC systems are geared towards supporting textual artifacts such as source code—managing them on a line-oriented level. In contrast, many software engineering artifacts including models are not managed on a line-oriented level, and thus a line-oriented change management is not adequate. For example, adding an association between two classes in a UML class diagram is a structural change, which is neither line-oriented, nor should be managed in a line-oriented way. However, a single structural change in the diagram is managed as multiple line changes by traditional VC systems. Nguyen et al. describe this problem as the *impedance mismatch* between the flat textual data models of traditional VC systems and graph-based software models [5]. Different approaches have been proposed to cope with the shortcomings of existing methods and techniques to better support change tracking and versioning of graph-based models. They can be categorized into two different classes: state-based and change-based approaches [8].

State-based approaches only store states of a model, and thus need to derive differences by comparing two states, e.g. a version and its successor, *after* the changes occurred [8]. This activity is often referred to as differencing. The differencing process can be viewed as a calculation to derive the change post-mortem and is generally expensive in computation time.

Change-based approaches record the changes, *while* they occur, and store them in a repository. There is no need for differencing, since the changes are recorded and stored, and thus do not need to be derived later on. **Operation-based approaches** are a special class of change-based approaches which represent the changes as operations transforming a state of a model into its successor state [8].

Several publications exist that show advantages of change-based and in particular operation-based approaches over state-based approaches in use cases such as conflict detection and merging [2], [9], [10], repository mining [11], inconsistency detection [12], and coupled evolution [13], [14]. However, there are no results available on the advantages of operation-based approaches in the most common use case of a VC system: *reviewing and understanding change*. We claim that understanding change is the most important use case of a VC system from a user's point of view, as it is required for almost any other use case, e.g. commit, update, merge, etc. Therefore, we believe that it is essential to conduct experiments on how well this use case is supported by the state-based and operation-based approaches.

In this paper, we discuss representatives of the different

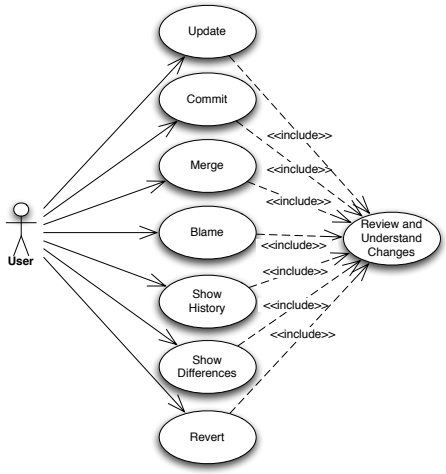

Figure 1. Use cases of a VC system (UML use case diagram)

approaches as well as the advantages and disadvantages of each type of approach in general. To *qualitatively* compare them, we present frequent use cases of a VC system and show how they are supported by the respective approach. Finally, we present the results of an empirical study we conducted to *quantitatively* compare a state-based with an operation-based approach for the purpose of reviewing and understanding change.

Outline. Section II introduces common use cases of a VC system. Section III compares the state-based approach and the operation-based approach to change tracking in a VC system. Related work is mentioned in the form of inline citations within these two sections. Section IV presents the design of the empirical study to compare both approaches, and Section V presents its results. Section VI concludes the paper with a short summary.

II. Use Cases of a Version Control System

A VC system has to fulfill a lot of use cases, many of which do not differ for state-based or change-based systems. Consider the use case of baselining, in which the user marks a certain approved version (e.g. a release). Since changes are not at all considered in this use case, both approaches appear identical from a user's point of view. Consequently, we only focus on use cases where a difference between state-based and change-based systems arises. We derived these use cases from well-known tools, such as the Revision Control System (RCS) [15], the Concurrent Versioning System (CVS) [16], and Subversion (SVN) [17], from research tools such as SiDiff [18] and UNICASE [19], and from the survey publications by Conradi and Westfechtel [8] and Dart [20]. Figure 1 illustrates the use cases that we consider important for discussing the differences between state-based and change-based change tracking. For every use case, we provide a short name and a description.

Update. The users retrieve changes between their local version and a target version (mostly the current head version) from the repository. These incoming changes can be reviewed by the user, before they are incorporated into the local working copy of the model. If the user accepts the changes, and they do not conflict with local changes, the version of the local working copy is set to the target version, and the changes are incorporated into the local copy.

Commit. The user decides to share the changes from their local working copy with the repository. The user reviews the changes before the commit to ensure that only intended changes are sent to the repository. If the user proceeds, the changes are sent to the repository to create a new version. In case the changes conflict with other commits that occurred since the last update, the commit is canceled by the VC system, and an update must occur first.

Merge. When incoming changes conflict with existing local changes in the update use case, the merge use case is initiated. The goal of a merge operation is to filter or transform the incoming and/or local changes, so that they do not conflict anymore. The result will be incorporated into the local workspace. The merging process involves manual work in most cases, requiring to review the changes. Merging may also occur if two branches in the repository are synchronized or rejoined, which essentially requires the same steps.

Blame. To find out how and by whom a problem or an inconsistency was created, the user is interested in finding recent changes on a certain part of the model. Typically, the last n changes on a selected set of model elements need to be retrieved. The user reviews these changes to find the change that causes the problem.

Show History. The user (often a project manager) reviews the history to get an impression of the current activities in a project. Mostly, the user is not interested in individual changes, but in an overview of how many and which type of changes on how many artifacts occurred.

Show Differences. The user is interested in reviewing the differences between two versions of a model, for example two releases. The two versions are typically not very close in terms of the number of changes between them.

Revert. The user wants to undo some changes in the local working copy. To ensure that the right changes are undone, they need to be reviewed beforehand.

Review and Understand Changes. The user reviews changes to understand what was changed, and most importantly, how it was changed.

Interestingly, the first seven use cases are placed most prominently in many VC systems and their clients [19], [16], [17]. We claim this is due to the fact that these are the most frequently executed use cases for the majority of users. *In all of these seven use cases, the user reviews changes in one or another way. As is shown in Figure 1, all use cases thus include the use case "Review and Understand Changes".*

III. Comparison of Change Tracking

In this section, we *qualitatively* compare different approaches to change tracking in a VC system based on a literature survey. Sections III-A and III-B introduce the state-based and operation-based approach, respectively. Section III-C contrasts the advantages and disadvantages of both approaches.

A. State-based Change Tracking

State-based approaches derive differences by comparing two states—e.g. a version and its successor—*after* the changes occurred. This activity is often referred to as differencing, and is performed in two phases: matching and comparison. In the matching phase, for each node in a certain state, the corresponding node in the other state is found. The matching can be based on the similarity of the node's content or on the graph structure it is connected to [21]. If the model supports unique identifiers, a matching does not need to be calculated, otherwise $O(n^2)$ comparisons are required for n nodes in a model [22], [23]. Chawathe et al. even claim that the matching problem for two states is NP-hard in its full generality [24]. In the comparison phase, each node is compared with its matching partner from the other state to derive potential changes. The comparison calculation requires $O(n)$. The space complexity for the whole differencing process is $2n$, since both states need to be present. In a state-based system, changes are not persisted in a way that reflects how they were actually performed. The differencing process can be viewed as a calculation to derive an approximation of the changes post-mortem.

Since the VC system is not required to be able to observe the changes while they occur, a total separation of the modeling tools and the VC system is possible. This is a clear advantage over change-based systems. It is even possible to use line-oriented VC systems and to perform differencing on the client side. However, the differencing tool must at least know the models' meta model, based on which the changes are calculated and represented. For example, EMF Compare [25] is a differencing tool for meta models defined with the meta modeling language Ecore of the Eclipse Modeling Framework (EMF) [26].

There are three main disadvantages of the state-based approach: (1) The temporal order of changes is lost, and it can not be perfectly derived. For understanding changes, the temporal order might be important. Moreover, the temporal order is useful for conflict detection and merging [9]. (2) Groupings of changes to composite changes are lost. Refactoring operations e.g. cause many changes that could be grouped. An example for a refactoring is an extract-superclass refactoring on UML class diagrams. This operation creates a new superclass for a number of existing classes and moves common attributes and associations to the new superclass. This results in many changes. Deriving composite changes—e.g. to detect refactorings—is difficult and in some cases even impossible due to masking problems [27]. (3) The computational complexity for differencing is high—especially if changes between many states need to be retrieved, or the model is of a large size [22], [23]. By means of the empirical study, the design of which is presented in Section IV, we want to find out whether and how the disadvantages (1) and (2) will affect the ability of users to understand change.

Considering the use cases presented in Section II, we make the following observations for the state-based approach:

Merge. The merge result can not be as accurate, since composite changes are not available [3], [9], [10]. Refactoring operations, for example, might only be partly reflected, if not all their caused changes are accepted.

Show History. The computational complexity for differencing could result in a severe performance problem, especially when looking at many versions and the changes that occurred in between them. Differencing is required for every such version.

Review and Understand Changes. We believe that disadvantages (1) and (2) are impacting the ability of humans to understand change. The temporal order of the changes, which is lost using state-based approaches, could help to understand the context in which the changes were performed. Composite changes could group changes that look unrelated, and thus have to be grouped in the user's mind.

B. Operation-based Change Tracking

In contrast to state-based approaches, change-based approaches record the changes *while* they occur, and store them in a repository. This implies that change-based systems persist changes in a way that reflects how they were actually performed. There is no need for differencing, since the changes are already available by design. Operation-based approaches are a special class of change-based approaches which represent the changes as transformation operations on a state [9]. An operation can be applied to a state of a model to transform it into the successor state [8].

Figure 2 shows the simplified taxonomy of operations from the UNICASE system [19], [3]. All operations refer to one ModelElement that is being changed by the operation and that is unambiguously identified by a unique identifier. A ModelElement is a node of the graph that can be linked to other nodes and that has values for a number of attributes. An AttributeOperation changes the value of an attribute of a model element. A ReferenceOperation creates or removes one or several links between model elements. A CreateDeleteOperation creates or deletes a model element. The created or deleted model element is contained in the operation in the case of operation-based change tracking. A CompositeOperation allows to group several related operations—e.g. to represent a refactoring.

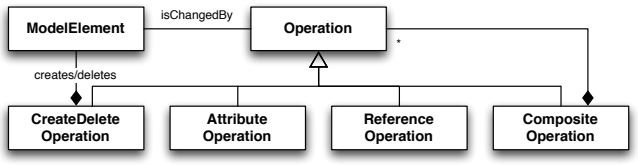

Figure 2. Taxonomy of operations (UML class diagram)

The change-based approaches have one disadvantage in common: they require the VC system to be present when the changes occur, i.e. when the modeling tool is manipulating the model. This requires an integration of the VC system into the modeling tool. However, this does not imply that the system must instrument the modeling tool, but may only use the infrastructure based on which the tool is built. For change recording, observer mechanisms can be used; for composite detection, the command pattern can be used. In case of EMF models, one can rely on the EMF notifications and command stack [26]. This effectively decouples the VC system from the modeling tool.

In general, change-based approaches can preserve the exact temporal order, in which the changes occurred. This is an important information for understanding changes, but is also useful to improve applications such as conflict detection and merging [9], [10]. Moreover, the exact times at which the changes occurred, can be recorded. Operation-based systems can record composite operations which express the fact that the contained operations occurred as an atomic unit. For example, an extract-superclass refactoring can be captured in a composite operation. This can help to understand changes, but is also helpful for conflict detection and merging [28], [10]. Since operations are essentially a command pattern with persistent commands, the operations can also be used to implement undo and redo functionality [10], [29]. Operation-based systems can provide a filter method to canonicalize a sequence of operations, which hides operations that are fully masked by later operations. For example, an extract-superclass refactoring is fully masked by a later deletion of all the involved classes.

Robbes et al. even claim that only an operation-based VC system allows for effective research on evolution, since it provides all the required information [11]. Moreover, many researchers rely on information from a VC system to derive quantitative data to evaluate their approaches. This technique is in wide-spread use and is commonly known as repository mining. For some approaches—for example recommendation for traceability links—it is not only necessary to be able to retrieve every version, but also to recover intermediate states in between two versions. For example, for a realistic simulation of a recommendation use case in a post-mortem analysis, it is necessary to restore the state just before the recommendation. This scenario can only be supported by operation-based VC systems, since the temporal order of changes is available.

Considering the use cases presented in Section II, we made the following observations for the operation-based approach:

Update. The operations incoming from the repository are presented to the user. If the difference between local and target version is large, the system can canonicalize the operations to get a more compact representation. Conflict detection can fully rely on the operations, their temporal order and composites to supply a more accurate result and avoid unnecessary conflicts [10]. In general, conflict detectors apply a conservative estimation: If they are unsure about a potential conflict, they raise the conflict to avoid later data corruption.

Commit. The recorded operations can be presented to the user, possibly after canonization. No differencing is required. Conflict detection may again profit from the additional information.

Merge. The merge can operate on the top level operations. Therefore, less decisions are needed, since many operations are contained in a composite operation. Moreover, the decisions can not partially mask a refactoring, as opposed to the state-based case [27].

Show Differences. This use case is best served by a state-based representation. It can be implemented directly by relying on an existing state-based approach or by deriving it from the recorded operations.

Review and Understand Changes. The changes can be presented as operations in the correct temporal order and can be grouped as composite operations.

C. Summary

State-based approaches exhibit the advantage that they are independent of the tool used for changing the models. State-based approaches derive an approximation of the exact change which is sufficient for certain kinds of changes and for certain use cases. However, the need to derive the changes from the states is a disadvantage of state-based approaches: Due to the graph isomorphism problem, calculating the difference is a computationally complex endeavor. Moreover, state-based approaches can neither completely and correctly derive the exact temporal order of the changes nor are they able to derive composite changes.

Operation-based approaches have the advantage that the changes are explicitly recorded. Therefore, no computation effort is necessary to derive the changes, when they are required for the different use cases. Moreover, operation-based approaches retain the exact temporal order of the changes as well as composite changes. Operation-based approaches exhibit the disadvantage that they need to be integrated into the tool used for changing the models. As a consequence, they cannot be used for existing tools which do not provide such a functionality.

The operation-based approach might seem very different from a state-based approach, but actually, it is only an enhancement. It records additional information that is lost in the state-based approach. In an operation-based approach, we can perform everything that can be done in a state-based approach—by just ignoring the additional information. This boils down to the question whether the additional effort for recording the changes is justified by its advantages. Therefore, we have conducted an empirical study to compare state-based with operation-based change tracking for the use case of reviewing and understanding change.

IV. Design of the Empirical Study

In this section, we present the design of the empirical study to *quantitatively* compare state-based and operation-based change tracking. Section IV-A lists the research questions that underlie the empirical study. Section IV-B presents the tools we chose as representatives for state-based and operation-based change tracking. Section IV-C describes the data we used as input for the evaluation. Section IV-D enumerates the different steps we carried out to conduct the empirical study.

A. Research Questions

We conduct the empirical study to answer the following research questions:

1) **Do users better understand the changes in a state-based or an operation-based representation?**
2) **Which factors influence the user in understanding a state-based or an operation-based representation?**

B. Setup

State-based change tracking is represented by the open-source tool EMF Compare [25], which is the state-of-the-art differencing and merging implementation for EMF (Eclipse Modeling Framework) [26]. It is delivered with the Eclipse Modeling Tools, which is one of several official Eclipse products. As we do not want to disadvantage EMF Compare in the study, the used matching strategy is based on unique identifiers to ensure correct matchings.

Figure 3 depicts an example of a change as represented by EMF Compare. The upper part shows the changes between two versions structured according to the target version. The lower part shows the target and the source version, respectively, and highlights affected elements. When a change is selected in the upper part, the affected elements are automatically selected in the lower part.

Operation-based change tracking is represented by the open-source CASE tool UNICASE [19], which is based on a unified model. It consists of a set of editors to manipulate instances of a unified model, and a repository to persist and version the model as well as to collaborate on the model. The unified model covers the whole development process from requirements over design to deployment, including

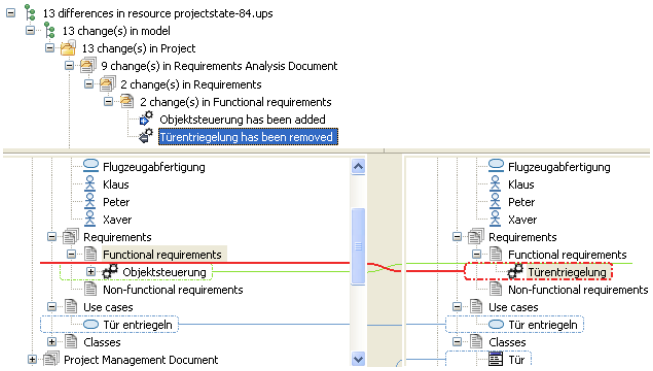

Figure 3. State-based representation (EMF Compare)

project management artifacts. System model elements such as requirements or UML elements, are part of the same unified model and stored in the same repository as project model elements such as tasks or users. UNICASE is implemented based on EMF, and realizes operation-based change-tracking, conflict detection and merging [3].

Figure 4 depicts the example change as represented by UNICASE. The representation shows the sequence of operations which have been executed on the source version to get to the target version. Note that the temporal order is from bottom to top. For each operation, its affected elements are shown below the operation. When an affected element is selected in the operation-based representation, it is automatically selected in a view of the target version (which is not shown in the figure).

C. Input

UNICASE was used to record operation histories, which we use as an input to the empirical study. The version model (see Figure 5) of UNICASE is a tree of versions with revision links [4]. Every version contains a change package and may contain a full version state representation. A change package contains all operations that transformed the previous version into this version along with administrative information such as the modifying user, a time stamp and a log message.

UNICASE was employed in a project named DOLLI2 (Distributed Online Logistics and Location Infrastructure 2) at a major European airport. The objective of DOLLI2 was integrating facility management and telemetry data into the tracking and locating infrastructure developed in the previ-

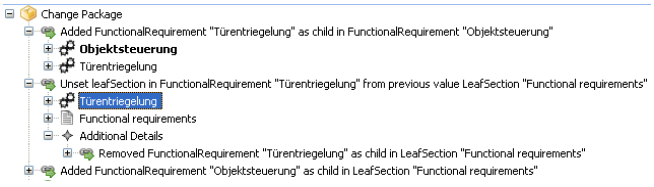

Figure 4. Operation-based representation (UNICASE)

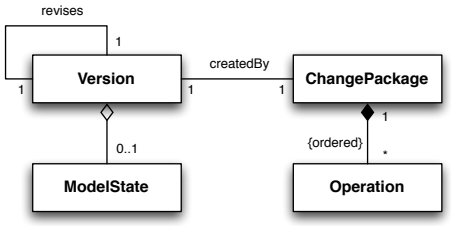

Figure 5. Version Model (UML class diagram)

ous project, together with expanding the 3D visualization on desktop computers as well as porting it to mobile devices. More than 20 student developers worked on the project for a period of five months with the airport as the industrial customer. All modeling was performed in the UNICASE tool. This resulted in a comprehensive model and a history of over 600 versions. At the head version, the model consists of 1684 model elements, which are from the following model types: 385 - task, 41 - organization, 251 - requirement, 299 - UML class, 32 - document, 281 - rationale, and 383 - meeting.

UNICASE also provides an Empirical Project Analysis Framework (EPAF). Iterators can be reused to run through all revisions of a model in a predefined way. Analyzers are used to analyze and extract data per revision, and exporters write the data to files. We used this framework to retrieve and analyze the data from the UNICASE VC system.

D. Conducting the Empirical Study

We apply the following process to conduct the empirical study, which consists of two phases. In the preparation phase, we randomly select a number of commits from the repository. In the interview phase, we present these commits in different representations to a number of users.

Preparation phase. This phase consists of the following steps:

Step 1: Mine the repository. We query the UNICASE VC system for all commits of the DOLLI2 project and extract the project state after the commit as well as the change package of the commit. Also we preserve for any version the state of its predecessor version to be able to create a state-based difference in the following data extraction. For each commit, we additionally record the following data: (a) The *commit size* is measured as the number of primitive changes. (b) The *commit complexity* is measured by a dependency depth value. This value is supposed to measure how much the changes of one commit depend on each other. Operation a *requires* operation b means that "a cannot be applied to a model without b" [3]. The *dependency depth* is the longest path of *requires* relations in an operation sequence. (c) The *commit category* determines what kind of operations a commit contains: Category 1 contains only AttributeOperations and ReferenceOperations, category 2 also contains CreateDeleteOperations, and category 3 also contains CompositeOperations (for the taxonomy, see Figure 2).

Step 2: Choose Users. We choose a number of users which are familiar with the input as well as a number of users which are not familiar with the input. We record for all users whether they are familiar with the input using the attribute *internal* for every data set.

Step 3: Extract data for users. For each user, we randomly select 18 commits of a total of 700 commits from the operation-based repository. We select only commits with a size greater than 5 and smaller than 30. We exclude the shorter commits, as we do not expect any difference between both representations. We exclude the longer commits, as understanding them would take too much time in the interview.

For each commit, we randomly decide whether the user is shown either the operation-based or state-based representation. Also we randomly determine the order in which the commits are presented to the user. We only take one representation for the same commit, as the first representation might ease the understanding of the second representation. Moreover, we ensure that each user is presented roughly the same number of commits in operation-based and state-based representation. To be able to correlate the answers with the complexity of the contained operations, we randomly sample 6 commits for each of the three above-mentioned categories. For each user, there is a so-called *shadow* user which is presented the same commits in the opposite representation. We generate the state-based representation for all the sample commits using EMF Compare. For the operation-based representation we just store a change package from the UNICASE VC system.

Interview phase. An interview is limited to a total of one hour per user including a short training on the different representations. Since the order of the shown commits is random, the interview can be ended at any time. If the interviewee completed the interview on all 18 data sets in less than one hour, the interview was also stopped. In the *interview phase*, we performed the following steps for each user and each commit:

Step 1: Present the commit to the user. Based on the data extracted before, we present the commit to the user in a state-based or an operation-based representation. The user is shown the representation in the respective tool which can be used to navigate through the changes and the project (see Figures 3 and 4). The user should do their best to understand the changes within a given time limit of 2 minutes. We determined the time limit by experimenting with several test subjects. The time limit is supposed to prevent the user from memorizing the changes rather than understanding them.

Step 2: Question the user about the commit. We assess the understanding of the user by means of exam questions. Once the user decides to take the exam or once the time for understanding the changes is up, the users can no longer

look at the change representations. In other words, the exam is closed book. The exam consists of the following question types: (a) *Understand the impact of the changes*: The user is confronted with a randomly selected element—that was changed by the commit—in two versions: the version before and the version after the commit. The user should try to answer the following question: "Which is the version of the element after the commit?" Since elements that were deleted or created by the commit are not present in both versions, we show a different question on deleted or created elements: "Was this element created, deleted, neither deleted nor created?" We generate 5 instances of the first question type and 2 of the latter, if the commit has enough elements to generate as many questions. The questions are supposed to determine whether the user has truly understood the impact of the presented changes. (b) *Understand the overall intention of the changes*: The user is presented with 10 commit messages and is asked to select the message that is assigned to the commit. The other messages are randomly selected from all commits of the respective project. Duplicates are removed. Out of these 10 messages, the user can select and prioritize a maximum of three candidates for the message of the presented commit.

Collected metrics. During the interview, we recorded the following metrics that can be used for statistical evaluation:

- the *time taken* for understanding the changes, which is between 0 and 2 minutes.
- the *compare score* based on the exam questions of type (a). The compare score is the sum of the evaluation of all questions divided by the number of questions. A question evaluates to 1, in case it was answered correctly, and to -1 otherwise. As a consequence, the aggregated result also is in the range of -1 and 1.
- the *log message score* based on the exam questions of type (b). The user obtains 4 points, if her first candidate is the correct commit message, 2 points for the second, 1 point for the third, and 0 points if none of her candidates is correct.
- the *self assessment* of the user reflecting the difficulty she felt in understanding the changes. For this measure, we use a scale with the following five values: very difficult (=1), difficult (=2), OK (=3), easy (=4), very easy (=5).

V. Result of the Empirical Study

In this section, we present the results of the empirical study. Section V-A evaluates the results by means of statistical tests. Section V-B interprets the results in terms of the research questions. Section V-C lists threats to the study's validity along with their mitigation.

A. Evaluation

We measured 162 change assessments from 14 different users. To evaluate these measurements, we performed a number of statistical tests [30]. An overview of all measurements is shown in Table I and II, the full evaluation results are available for download on a companion webpage [31].

Table I
Average Values for each Metric per User

User	Prof.	Internal	Size	Complexity	Compare	Log	Time	Self
a	PhD	yes	11.86	1.71	0.65	1.43	53.99	4.29
b	PhD	yes	11.86	1.71	0.59	3.14	63.07	4.29
c	BSc	yes	15	0.5	0.4	2	136.99	3.5
d	BSc	yes	14.67	0.58	0.58	2.5	86.01	2.5
e	BSc	yes	12.17	0.42	0.76	3.17	58.82	4.42
f	BSc	yes	13.5	0.3	0.77	3.3	89.23	3.7
g	PhD	no	11.08	1.87	0.77	2.54	87.56	4.54
h	PhD	no	10.8	1.8	0.75	3.2	117.16	3.6
i	MSc	no	14.94	2.67	0.55	2.72	61.79	2.78
j	MSc	no	13.67	3.17	0.63	0.67	57.31	3.75
k	MSc	no	12.27	3.09	0.18	1.45	90.25	2.91
l	PhD	no	17.08	2.92	0.7	2	118.88	3.67
m	BSc	no	11.62	1.92	0.6	2.46	86.59	2.85
n	BSc	no	12.18	1.56	0.64	2.18	54.39	4

Prof.: PhD student, MSc student and BSc student, all majors Computer Science related
Internal: yes if the user is familiar with the input data
Size/Complexity: average size/complexity of the commits
Compare: average compare score, range is [-1,1], 1 is best
Log: average log message score, range is [0,4], 4 is best
Time: average time in seconds that the user spent on reviewing each commit
Self: average self-assessment, range is [1,5], 5 is best

Table II
Average Values for Metric per Representation

Representation	Compare	Log	Time	Self	Compare (Simple)	Compare (Complex)
State-based	0.65	2.37	78.07	3.71	0.74	0.46
Operation-based	0.58	2.32	83.45	3.41	0.73	0.62

Columns 2-5: average measurement values for each metric and representation
Compare (Simple): average compare score for category 1 commits with size < 10
Compare (Complex): average compare score for category 2 and 3 commits with size ≥ 10

To get a first glance of the understanding process of the user when confronted with the two representations, we divided the commits into two groups: those which were state-based ($n_1 = 75$ items) and those which were operation-based ($n_2 = 87$ items). We chose the variable *compare score* and *log message score* as key variables, since they describe to which extent the user has correctly understood the performed change. Our hypothesis was that the operation-based representation should be more understandable, i.e. that the means of the *compare score* and/or *log message score* differ significantly in both groups. Assuming a normal distribution of both values, we applied the T-test [30] for independent samples to check the hypothesis of equal mean values. However, the test results showed that there was no significant difference between the two groups for neither variable. Then we applied the T-test for *taken time* and *self-assessment*—also ending up with no significant result.

Based on the advantages of the operation-based representation, we expected that users should better understand more complicated commits in the operation-based representation. As described earlier, we have recorded different measures for complexity: commit size, complexity and category. We

decided to narrow down the commit samples to the commits with higher complexity according to the provided measures. We tried all measures as decision criteria individually and in combination with others. The statistical results showed that the most significant difference between state- and operation-based representation are in the group of complex changes from category 2 and 3—given a commit size greater than or equal to 10 (see Table II for average values).

In the following, we detail the statistical results of this group. We separated the filtered commits into two subgroups: those which were state-based ($n_1 = 34$ items) and those which were operation-based ($n_2 = 37$ items). Our hypothesis was that the means of the *compare score* differ significantly in both groups. Assuming a normal distribution of both values, we applied the T-test for independent samples to check the hypothesis of equal mean values. The 95%-confidence interval for the difference of the mean values $m_{state} - m_{op} = -0.195$ is $[-0.389, -0.002]$. The T-test returned a T-value of -2.019, which means that the critical value $-c(df = 60, \alpha = 5\%) = -1.67$ for 60 degrees of freedom is exceeded according to amount and thus the null-hypothesis has to be rejected on the 5% level of significance. Thereby, the variances of both groups cannot be assumed to be equal, since the F-test [30] returned an F-value of 4.0, which is greater than the critical value $c(df_1 = 34, df_2 = 37, \alpha = 5\%) = 1.74$ for that level of significance and the given degrees of freedom. Thus we used the T-test for different variances in both groups, implying a lower number of degrees of freedom—only about 60, in contrast to $n_1 + n_2 - 2 = 69$ (two degrees are subtracted, since the expected value and the variance have to be estimated from the data sample). The variance of the user assessment regarding the operation-based representation is significantly lower than for the state-based representation. This might argue for a higher robustness of the operation-based representation, since the users are more consistent in their assessment of the changes—as illustrated by Figure 6. Besides the discriminating variable *compare score*, we also performed tests on other variables, i.e. *taken time*, *log message score* and *self-assessment*, but no substantial difference was found. Especially, the distribution of the binary variable *internal*, which indicates whether the respective change was assessed by a member of the development team or by an external person (see IV-D), did not significantly differ within the two groups of state-based and operation-based changes.

Even if one argues that the number of changes in each group is not normally distributed, the non-parametric Mann-Whitney-U test [30] can be used to recheck the result. Due to its non-parametric character, the Mann-Whitney-U test is weaker than the T-test, but still succeeds in rejecting the null-hypothesis of equal means at the 10% level of significance. The test returns a Z-value of $Z = -1.739$ (standard-normal approximation), which makes the probability to erroneously reject the null-hypothesis relatively small:

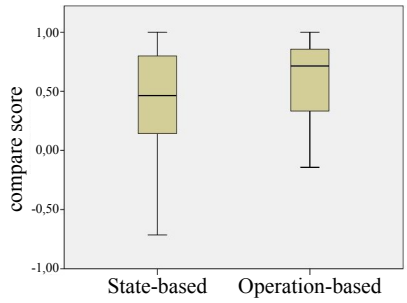

Figure 6. Boxplot of the compare score distribution of state-based and operation-based representation

$\Phi(Z) + (1 - \Phi(-Z)) \overset{symm.\ of\ \Phi}{=} 2 \cdot \Phi(Z) \approx 0.082$ (asymptotical significance).

B. Interpretation

In this section, we interpret the results in terms of the research questions posed in Section IV-A.

Do users better understand the changes in a state-based or an operation-based representation? The statistical result shows no significant overall difference between the respective representations in the variables *compare score*, *log message score*, *taken time* and *self-assessment*. In general, we observed that the *log message score*, *taken time* and *self-assessment* mostly showed a similar distribution in both representations. The *compare score* often deviated—but below a reasonable significance level. We already supposed that the operation-based approach might provide better results with increasing complexity of the changes. Therefore, we have also recorded different measures for complexity for each commit. The results showed that by partitioning the data sets according to category and size, we obtain a set of more complex commits, where the operation-based representation is significantly better than the state-based representation. By design, the *compare score* required a more detailed understanding of the changes, while the *log message score* required a birds-eye-view understanding. We believe this is why only *compare score* shows significant differences in both representations. The log message can often be guessed—e.g. by memorizing frequent words—which is equally well supported in both representations.

With respect to the research question, we conclude that users better understand changes in an operation-based representation, if the changes are sufficiently complex and an in-depth understanding is required. Also, we conclude that in the case of less complex changes, neither representation shows big advantages, since no significant difference in the variables' distribution could be determined. We believe that the reason for the advantage of the operation-based representation for complex changes is that operations represent complex changes composed of many simple changes more compactly and therefore in a more readable way.

Which factors influence the user in understanding change in a state-based or an operation-based represen-

tation? Based on the results from grouping the commits by the measures introduced above, we made the following observations on factors relevant for understanding the change. The *dependency depth* does not seem to be relevant, since its individual and combined use for grouping did not improve significance for any of the variables for neither representation. The *category* however seems to be a relevant factor. Both category 2 and 3 only contain changes that are composed of potentially many atomic changes. A delete operation for example removes an element and its children along with all cross references targeting them from the model. Using commits only from these categories, already showed promising results—however, below a reasonable significance level. Only when combined with a restriction of the *commit size* (≥ 10), significant differences appeared. In our opinion, the reason for this observation is that a commit with complex operations—but of a size of less than 10 primitive changes—is not very difficult to understand in neither representation. We conclude that commit size and category are factors influencing the understanding of the change in the two given representations.

C. Threats to Validity

We are aware of the following internal and external threats to validity, which might have affected our results.

Internal Validity. The results might be influenced by the design we chose for the empirical study. Thereby, the results might be affected by the way we prepared the data and performed the interviews.

Commit selection. When preparing the data for the evaluation, we might have chosen commits which favor one representation over the other. Consequently, one approach might perform better than the other, which poses a threat to the result's validity. To mitigate this threat, we randomly chose the commits and sampled them according to different categories of changes.

Understanding vs. memorization. One could argue that the users only need to fully memorize the commit for answering the questions. As a consequence, we would be examining the user's ability to memorize a number of changes instead of their ability to understand the changes. To force the users to really understand the commit, we limited the time for looking at each commit.

User Interface Usability. The usability of the respective tools could be an influencing factor. The effort put into the development of the operation-based research prototype is orders of magnitude less than for EMF Compare. So we believe—if there is a usability difference—it is rather a disadvantage for the operation-based approach. To mitigate this threat, we trained the users in the usage of the tools and ignored results when a user interface problem with the respective tool occurred.

External Validity. The results might be influenced by the fact that we questioned only a restricted number of users about a restricted number of commits. Thereby, the results might not be representative for all developers or for all possible model changes on average.

Prior knowledge. The users might have prior knowledge about a certain representation which favors the respective representation. This poses a threat to the transferability of the results to all users in general. To mitigate this threat, we trained all participants of the empirical study on a number of samples for both representations in advance.

Specific modeling language. The results might be affected by the modeling language of UNICASE in which the models are developed. As a consequence, the results might not be transferable to the evolution of models created with other modeling languages. However, the UNICASE modeling language is similar to UML, whose different dialects are widely used in software engineering practice.

Small Population. The number of participants is only 14 users. However, the results for the different users do not show significant differences and thereby the more important number is the number of commits, which is 162.

VI. Conclusion and Future Work

Models are an important artifact in a software development project. There are different approaches to support collaboration on and versioning of models. We reviewed both state-based and operation-based approaches. We compared both approaches by means of typical use cases of VC systems in a qualitative way. We also conducted a case study to compare the two approaches in the use case of reviewing and understanding changes in a quantitative way. The results from our case study indicate that operation-based change tracking exhibits advantages in understanding more complex changes. In addition, we found the size and type of changes of a commit to be relevant factors for understanding changes. Based on these results, we believe that the additional information recorded by the operation-based approach can be valuable in this central use case. Further research is required to find further determining factors and to decide in which use case and under which conditions which representation should be used.

VII. Acknowledgments

The presented work was partially funded by the German Federal Ministry of Education and Research (BMBF), grant "SPES2020, 01IS08045A".

References

[1] C. Bartelt, "Consistence preserving model merge in collaborative development processes," in *CVSM '08: Proceedings of the 2008 international workshop on Comparison and versioning of software models*. New York, NY, USA: ACM, 2008, pp. 13–18.

[2] D. Dig, K. Manzoor, R. Johnson, and T. N. Nguyen, "Refactoring-aware configuration management for object-oriented programs," in *ICSE '07: Proceedings of the 29th international conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 427–436.

[3] M. Koegel, J. Helming, and S. Seyboth, "Operation-based conflict detection and resolution," in *CVSM '09: Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 43–48.

[4] M. Kögel, "Towards software configuration management for unified models," in *CVSM '08: Proceedings of the 2008 international workshop on Comparison and versioning of software models*. New York, NY, USA: ACM, 2008.

[5] T. N. Nguyen, E. V. Munson, J. T. Boyland, and C. Thao, "An infrastructure for development of object-oriented, multi-level configuration management services," in *ICSE '05: Proceedings of the 27th international conference on Software engineering*. New York, NY, USA: ACM, 2005.

[6] D. Ohst, "A fine-grained version and configuration model in analysis and design," in *ICSM '02: Proceedings of the International Conference on Software Maintenance (ICSM'02)*. Washington, DC, USA: IEEE Computer Society, 2002, p. 521.

[7] J. Rho and C. Wu, "An efficient version model of software diagrams," in *APSEC '98: Proceedings of the Fifth Asia Pacific Software Engineering Conference*. Washington, DC, USA: IEEE Computer Society, 1998, p. 236.

[8] R. Conradi and B. Westfechtel, "Version models for software configuration management," *ACM Comput. Surv.*, vol. 30, no. 2, pp. 232–282, 1998.

[9] E. Lippe and N. van Oosterom, "Operation-based merging," in *SDE 5: Proceedings of the fifth ACM SIGSOFT symposium on Software development environments*. New York, NY, USA: ACM, 1992, pp. 78–87.

[10] T. Mens, "A state-of-the-art survey on software merging," *IEEE Trans. Softw. Eng.*, vol. 28, no. 5, pp. 449–462, 2002.

[11] R. Robbes, M. Lanza, and M. Lungu, "An approach to software evolution based on semantic change," in *Fundamental Approaches to Software Engineering*, ser. Lecture Notes in Computer Science, vol. 4422. Springer Berlin / Heidelberg, 2007, pp. 27–41.

[12] X. Blanc, I. Mounier, A. Mougenot, and T. Mens, "Detecting model inconsistency through operation-based model construction," in *ICSE '08: Proceedings of the 30th international conference on Software engineering*. New York, NY, USA: ACM, 2008, pp. 511–520.

[13] M. Herrmannsdoerfer, S. Benz, and E. Juergens, "COPE - automating coupled evolution of metamodels and models," in *ECOOP 2009 - Object-Oriented Programming*, ser. Lecture Notes in Computer Science, vol. 5653. Springer Berlin / Heidelberg, 2009, pp. 52–76.

[14] G. Wachsmuth, "Metamodel adaptation and model co-adaptation," in *ECOOP 2007 - Object-Oriented Programming*, ser. Lecture Notes in Computer Science, vol. 4609. Springer Berlin / Heidelberg, 2007, pp. 600–624.

[15] W. F. Tichy, "Design, implementation, and evaluation of a revision control system," in *ICSE '82: Proceedings of the 6th international conference on Software engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1982, pp. 58–67.

[16] D. Prince, "CVS," http://www.nongnu.org/cvs.

[17] Tigris, "Subversion VC System," http://subversion.tigris.org.

[18] U. Siegen, "SiDiff," http://sidiff.org.

[19] J. Helming, M. Koegel, "UNICASE," http://unicase.org.

[20] S. Dart, "Spectrum of functionality in configuration management systems," CMU/SEI, Tech. Rep., 1990.

[21] D. S. Kolovos, D. Di Ruscio, A. Pierantonio, and R. F. Paige, "Different models for model matching: An analysis of approaches to support model differencing," in *CVSM '09: Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–6.

[22] T. Lindholm, J. Kangasharju, and S. Tarkoma, "Fast and simple xml tree differencing by sequence alignment," in *DocEng '06: Proceedings of the 2006 ACM symposium on Document engineering*. New York, NY, USA: ACM, 2006.

[23] C. Treude, S. Berlik, S. Wenzel, and U. Kelter, "Difference computation of large models," in *ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. New York, NY, USA: ACM, 2007, pp. 295–304.

[24] S. S. Chawathe and H. Garcia-Molina, "Meaningful change detection in structured data," in *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 1997.

[25] Eclipse, "EMF Compare," http://wiki.eclipse.org/EMF_Compare.

[26] ——, "Eclipse Modeling Framework," http://www.eclipse.org/emf.

[27] Z. Xing and E. Stroulia, "Refactoring detection based on umldiff change-facts queries," in *WCRE '06: Proceedings of the 13th Working Conference on Reverse Engineering*. Washington, DC, USA: IEEE Computer Society, 2006.

[28] A. Lie, R. Conradi, T. M. Didriksen, and E.-A. Karlsson, "Change oriented versioning in a software engineering database," in *Proceedings of the 2nd International Workshop on Software configuration management*. New York, NY, USA: ACM, 1989, pp. 56–65.

[29] R. Robbes and M. Lanza, "A change-based approach to software evolution," *Electron. Notes Theor. Comput. Sci.*, vol. 166, pp. 93–109, 2007.

[30] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, 2007.

[31] M. Koegel and Y. Li, "Evaluation Data Webpage," http://www1.in.tum.de/static/shared/studies/svo/.