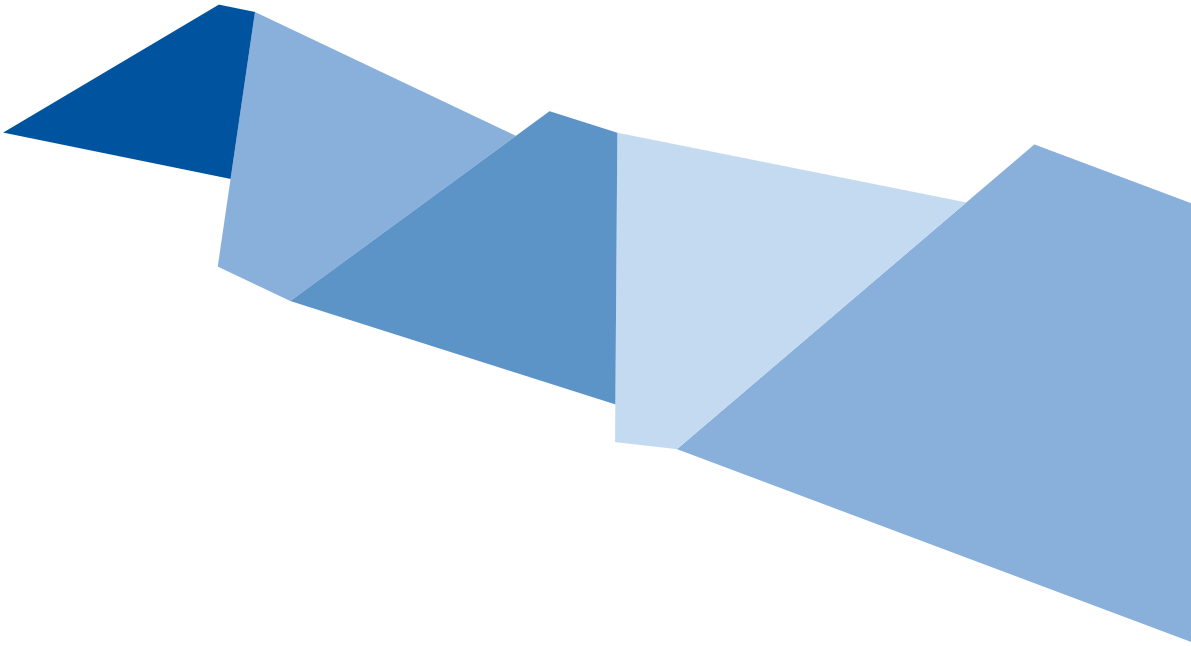


Pre-Study on the Usefulness of Difference Operators for Modeling Languages in Software Development

Imke Drave
Oliver Kautz
Judith Michael
Bernhard Rumpe

Department of Computer Science

Technical Report

Aachener Informatik-Berichte (AIB) | ISSN 0935-3232 | AIB-2020-05
RWTH Aachen University | Department of Computer Science | May 2020

The publications of the Department of Computer Science of *RWTH Aachen University* are in general accessible through the World Wide Web.

`http://aib.informatik.rwth-aachen.de/`

RWTH Aachen University
Software Engineering Group

Pre-Study on the Usefulness of Difference Operators for Modeling Languages in Software Development

Class Diagrams, Activity Diagrams, Feature Diagrams, and Statecharts

Imke Drave
Oliver Kautz
Judith Michael
Bernhard Rumpe

Project 'A Semantic Approach to Evolution Analysis in Model-Based Software Development (SemanticDiff)', 2015-2019, this research received funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project no. 250902306 - Geschäftszeichen: RU 1431/9-1.

Abstract

Models are the primary development artifacts in model-driven software engineering making model change management crucial for developers. In our work, we investigate if semantic differencing improves the developers' understandings of differences between model versions. Current research in this field focuses on pure syntactic differences between models which might not reveal the impact of the syntactic change to the real world. Thus, the semantic difference of models is an open field to investigate. We propose differencing operators for model comparison for four different modeling languages (Class Diagrams, Activity Diagrams, Statecharts and Feature Diagrams). This technical report describes the main fundamentals of the semantic differencing operators for the four modeling languages and the results of a pre-study on the usefulness of the differencing operators for software engineers. In the study, we were asking how well aspects such as syntactic differences, semantic differences, a combination of syntactic and semantic differences, and the abstraction as well as the summarization of semantic differences helped to understand the differences between shown models in each of the four modeling languages. The pre-study has shown that a combination of the syntactic and semantic difference is the most suitable alternative to providing intuitive explanations that take semantic differences into consideration. However, a larger study with real-size models is needed for obtaining more meaningful results.

Contents

Chapter 1

Introduction and Motivation

Since models are the primary development artifacts in model-driven software engineering (MDSE) [Sel03, Sch06, FR07, BCW17, KRR18], managing model changes is an important task [BKL+12] to support developers. Implementing effective change management is still a major and not yet fully addressed challenge in software engineering in general and in MDSE in particular and has attracted significant research attention in recent years, e.g., [OWK03, AP03, MGH05, OMW05, EPK06, XS07, MD08, ASW09, MRR11a, MRR11f, MRR12, GKLE13, GKLE13, KKT13, TELW14, TELW14, MR15, BKRW17, MR18, KR18b, KR18a, DKMR19, BKRW19, DEKR19]. Models continuously evolve during their design, development, and maintenance due to iterative development methodologies, changing requirements, and bug fixes. Thus, detecting and understanding the differences between model versions is crucial for developers.

Fundamental building blocks for the detection of model differences are differencing operators applicable for model comparisons [MRR11a]. Many works have investigated various kinds of model comparisons. Most of them present syntactic differencing operators focusing on comparing the syntactic elements of two models. A syntactic differencing operator takes two models as input and outputs a representation of syntactic model elements that must be removed, added, or changed in the one model to obtain the other model [AP03, EPK06, ASW09, XS07, GKLE13, BKL+12, KKT13, TELW14]. However, such syntactic differences might not reveal the essence of the differences: A truly refactored model exhibits syntactic differences to a previous version of the model, however, its meaning has not changed due to the syntactic transformation. Vice versa, two syntactically similar models may have very different semantics. Therefore, we have developed semantic differencing operators [KMRR17, BKRW17, KR18b, DKMR19, BKRW19, DEKR19] within the DFG-supported project SemanticDiff [1]. A semantic differencing operator takes two models as input and returns instances of the one model that are not instances of the other model [MRR11a]. These instances are diff witnesses that reveal those changes that cannot be detected by syntactic differencing operators.

Semantic differencing enables to display unintentional changes to a model's semantics, *i.e.,* the introduction of bugs. However, the way semantic (and also syntactic) differences are displayed to the developer has a strong influence on her understanding of the differences.

[1]Project 'A Semantic Approach to Evolution Analysis in Model-Based Software Development (SemanticDiff)', 2015-2019, this research is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project no. 250902306. - Geschäftszeichen: RU 1431/9-1

Often, developers conduct model differencing to find out whether a changed model still fulfills a certain property. If this is not the case, the pure semantic difference might not explain why the model does not fulfill the property intuitively. Techniques that identify the syntactic elements or syntactic changes that cause a semantic difference [KR18a] enable the developer to understand the relation between syntactic elements or syntactic changes and the semantic differences. Other presentation techniques summarize similar witnesses for semantic differences in order to present a more condensed and therefore understandable set of differences to the user [MRR11e].

To investigate how well semantic differencing techniques represent differences and help developers understand differences between model versions, we conducted a survey among software engineers and students. The survey investigates the effects of the representations of model differences on the comprehensibility of differences between model versions. Particularly, we have investigated the comprehensibility of five representations of differences between models of four different modeling languages, namely Class Diagrams (CDs), Activity Diagrams (ADs), Statecharts (SCs) and Feature Diagrams (FDs).

This technical report is structured as follows: The next chapter describes model differencing and especially introduces language-specific semantic differencing operators. Chapter 3 explains the design of our study in detail. Chapter 4 presents the results of the study and our main findings. Chapter 5 discusses related work. The last chapter concludes. Appendix A presents the questionnaire of the study.

Chapter 2

Model Differencing

Research on model differencing divides into syntactic differencing, semantic differencing, and approaches that combine syntactic with semantic differencing [MR15, MR18, KR18a].

Syntactic differencing operators focus on comparing the differences between syntactic elements of two models. Thereby, the focus lies on detecting those syntactic model elements that differentiate two model versions. A syntactic differencing operator takes two models as input and returns a set of syntactic changes, *i.e.,* the *syntactic difference*. A syntactic change could be the addition, removal or other manipulation, *e.g.,* renaming, of one or more model elements. The syntactic difference, thereby, encodes those syntactic changes that transform the one input model to the other [AP03, EPK06, ASW09, XS07, GKLE13, BKL$^+$12, KKT13, TELW14]. Syntactic differencing is not concerned with the semantics of models. Since syntactic differences do not imply semantic differences, syntactic differencing operators do not reveal the semantic differences between the two input models, *i.e.,* the instances of the one model that are not instances of the other. These usually represent the bugs fixed in a model or the features added to a model. To understand the semantic differences of models, developers have to interpret the syntactic differences which, without semantic differencing operators, is a manual task.

Semantic differencing operators focus on comparing the semantic differences of two models [MRR11a, MRR12, MR18, KR18b]. A semantic differencing operator takes two models as input and returns a finite set of *diff witnesses* [MRR11a, MRR11f, MRR12, MR15, MR18, KR18a]. A diff witness can be interpreted as an instance of the one model that is not an instance of the other model. Therefore, each diff witness is a concrete proof for the existence of a difference between the semantics of the input models. As semantic differencing abstracts from the syntactic differences, it may not provide an understandable representation of model differences that relates the syntax of the models to their semantics. To understand which syntactic differences cause a semantic difference, developers have to interpret the latter, which, without syntactic differencing, is a manual task.

Approaches combining syntactic and semantic differencing focus on relating the semantic differences to syntactic model elements causing their existence [MR15, MR18, KR18a]. The approach presented in [KR18a], detects syntactic changes that can be applied to a model in order to obtain a model that has no semantic differences compared to the other model [KR18a]. The approach interprets the model elements that are affected by the syntactic changes to cause the existence of the semantic differences. The computed syntactic changes can be directly applied to transform a model in case the semantic difference represents a bug.

Semantic differencing operators can be combined with abstraction techniques and with summarization techniques [MRR11f]. Abstraction techniques facilitate users in executing semantic differencing while abstracting from semantic differences caused by selected model elements. Summarization techniques condensate the information presented to developers by semantic differencing operators. Summarization techniques summarize similar diff witnesses and present only a single witness of multiple similar witnesses to developers.

In the following, Section 2.1 presents a formal framework for model differencing that is the basis of our survey. The framework formalizes the notion of modeling language, syntactic differencing, semantic differencing, summarization, abstraction, and presents a language-independent combination of syntactic with semantic differencing. Subsequently, Section 2.2 presents the instantiations of the framework with the modeling languages.

2.1 A Formal Framework for Model Differencing

The basis for all model evolution analyses is a precise definition of the modeling language [Rum98, HR00, HR04, CGR09, Grö10, GR11].

Definition 1 *A modeling language is a tuple* (M, S, sem) *where*

- M *is a countable set of models,*
- S *is a set called semantic domain, and*
- $sem : M \to \wp(S)$ *is a semantic mapping.*

In the following, let $\mathcal{L} = (M, S, sem)$ be an arbitrary but fixed modeling language. Unless stated otherwise, the following notions are defined relative to $\mathcal{L}$.

The set M contains all models that are syntactically correct with respect to some syntax definition. The definition of modeling language abstracts from the technology used to define the syntactic representation of models. The semantic domain S is a set of well-understood (mathematical) constructs. The elements of the semantic domain are interpreted as possible realizations of the modeled circumstances. The semantic mapping sem maps each model $m \in M$ to its meaning $sem(m)$. The set-based semantic mapping reflects that models are often highly underspecified in the sense that they permit multiple possible realizations. For example, the semantic domain for a CD modeling language can be defined as the set of all possible object structures. Then, the semantic mapping of the CD language maps each CD to the set of all object structures permitted by the CD.

2.1.1 Syntactic Differencing

The definition of modeling language captures the set of all models and the semantics of models but does not formalize changes to a model. Change operations are an adequate formalism to specify the possible syntactic transformations applicable to models in the language [MR15, MR18, KR18a].

Definition 1 *A change operation for* $\mathcal{L}$ *is a partial function* $o : M \rightharpoonup M$.

Change operations are partial functions because not every change applied to any well-formed model yields a syntactically correct model [KKT13, TELW14, GKLE13, MR15, MR18, KR18a]. An evolution step consists of applying multiple change operations to a model. A countable set O of change operations for $\mathcal{L}$ is called a *change operation suite* for $\mathcal{L}$ [KR18a]. Change sequences are sequences of change operations and *e.g.,* describe the histories of model versions or steps for systematically changing a model. A finite sequence of change operations is called a *change sequence* [KR18a]. For a set of change operations O, the set of all possible change sequences is denoted O^* and the length of a change sequence $t \in O^*$, *i.e.,* the number of change operations in t, is denoted $|t|$.

The operator $\triangleright\colon M \times O^* \rightharpoonup M$ is the point-wise extension of the application of change operations to the application of sequences of change operations. The extension is naturally defined by (1) $m \triangleright \varepsilon = m$, (2) $m \triangleright o : t = o(m) \triangleright t$, if o is defined on m, (3) $m \triangleright o : t$ is undefined, if o is undefined on m. By means of a change operation suite for a language $\mathcal{L}$, the language formally occupies a set of possible syntactic model transformations. Syntactic differencing requires the ability to change every model in M to every other model in M: A change operation suite O for $\mathcal{L}$ is said to be *complete* iff $\forall m, m' \in M : \exists t \in O^* : m \triangleright t = m'$ [KR18a]. By [MR15, MR18, KR18a], a change sequence t is called a *syntactic difference* from a model $m \in M$ to a model $m' \in M$ iff $m \triangleright t = m'$. In the following, O denotes an arbitrary but fixed complete change operation suite for $\mathcal{L}$.

Syntactic differencing research focuses on (automatically) deriving complete change operation suites and on computing syntactic differences between models. A syntactic differencing operator is a function $\Delta : M \times M \to O^*$ such that $\forall m, m' \in\colon m \triangleright \Delta(m, m') = m'$. Thus, a syntactic differencing operator takes two models as input and outputs a change sequence such that applying the sequence to the first model yields the second model. Syntactic differencing operators are, in general, not unique. Often, syntactic differencing operators are required to compute short(est) syntactic differences for the input models.

2.1.2 Semantic Differencing

The semantic domain S represents all possible realizations of the models in M. The semantic mapping *sem* maps each model to its set of possible realizations. This interpretation of the semantic domain and the set-based semantic mapping enable to easily define a refinement relation on models: Intuitively, a model is a refinement of another model iff every realization of the former is also a realization of the latter. Formally, this relation is established via set inclusion over the models' semantics [HKR$^+$07].

Definition 2 *A model $m \in M$ is a refinement of a model $m' \in M$ iff $sem(m) \subseteq sem(m')$.*

Thus, if every realization of a model is correct with respect to some requirements on the semantics of the model, then every realization of every refinement of the model is also correct with respect to these requirements. However, in most cases, a model does not refine another model. Then, the semantic difference [MRR11a] from the former model to the latter model is not empty.

Definition 3 *The semantic difference from a model $m \in M$ to a model $m' \in M$ is defined as the set $\delta(m, m') = sem(m) \setminus sem(m')$.*

The elements in the semantic difference $\delta(m, m')$ are called diff witnesses (for the semantic difference from $m \in M$ to $m' \in M$) [MRR11a]. Every diff witness represents a realization of the model m that is not a possible realization of the model m'. By definition, a model $m \in M$ is a refinement of a model $m' \in M$ iff the semantic difference from m to m' is empty, i.e., $\delta(m, m') = \emptyset$.

A *semantic differencing operator* [MRR11a] is an automatic procedure that takes two models $m, m' \in M$ as input and returns a finite set of diff witnesses $W \subseteq \delta(m, m')$ that are contained in the semantic difference from m to m'. Developers can survey the output of a semantic differencing operator to increase their understandings of the semantic differences between the two input models. A semantic differencing operator $\text{diff} : M \times M \to \wp(S)$ is applicable for automatic refinement checking if for all models $m, m' \in M$, it holds that $\text{diff}(m, m') = \emptyset$ iff $\delta(m, m') = \emptyset$. Research on semantic differencing focuses on developing language-specific semantic differencing operators.

2.1.3 Summarization in Semantic Differencing

Summarization techniques [MRR11f] condensate the information presented to developers by summarizing similar witnesses. Multiple witnesses might represent similar information concerning the semantic difference between two models. The idea of summarization techniques is to partition diff witnesses into equivalence classes. Each equivalence class should be chosen such that it contains witnesses that represent similar information concerning the semantic difference of the models. The underlying assumption is that presenting multiple witnesses from the same equivalence class does not provide more information to a developer than presenting one witness of the equivalence class.

Definition 4 *Let $\sim$ be an equivalence relation on S. A summary of the semantic difference from $m \in M$ to $m' \in M$ concerning $\sim$ is a set $Sum \subseteq \delta(m, m')$ such that $\forall w, w' \in Sum : w \neq w' \Rightarrow w \not\sim w'$.*

Independent of the used equivalence relation, if a semantic difference is not empty, there exists a non-empty summarization of the difference concerning the equivalence relation. For a set of diff witnesses contained in the semantic difference from a model to another model, Algorithm 1 computes a summary of the semantic difference that is a subset of the set of witnesses. An elementary operation of the algorithm is to check whether two witnesses are equivalent with respect to the equivalence relation $\sim$. The algorithm initializes the variable Sum as the empty set (l. 1). Then, the algorithm iterates over all witnesses contained in the set W (l. 2-6). For each of these witnesses w, the algorithm checks whether the set Sum already contains another witness that is equivalent to the witness w (l. 3). If this is not the case, then the algorithm adds the witness w to the set Sum (l. 4). Finally, the algorithm returns the computed summary (l. 7).

2.1.4 Abstraction in Semantic Differencing

Abstraction techniques perform semantic differencing on different levels of abstraction. The idea is to execute the semantic differencing operator while abstracting from selected syntactic model elements. Thereby, abstraction techniques support developers in detecting the syntactic model elements that are responsible for the existence of semantic differences.

Algorithm 1 Computing a summarization of a set of witnesses concerning an equivalence relation $\sim$ on S.

Input: A set of witnesses $W \subseteq S$.
Output: A summary of W concerning $\sim$.

```
1: define Sum ← ∅ set of S
2: for all w ∈ W do
3:     if ∀w' ∈ Sum : w ≁ w' then
4:         Sum ← Sum ∪ {w}
5:     end if
6: end for
7: return Sum
```

For the application of an abstraction technique we assume that a set of model elements $\mathcal{M}$ is given and that each model $m \in M$ can be considered as a finite set of model elements $m \subseteq \mathcal{M}$. For instance, for a CD modeling language, adequate modeling elements are classes, enumerations, associations, cardinalities, etc.

A semantic differencing operator that supports abstraction takes two models $m, m' \in M$ and a set of model elements $E \subseteq \mathcal{M}$ as input. If $m \setminus E \in M$ and $m' \setminus E \in M$ are well-formed models of the language, then the operator returns the result from applying the ordinary semantic differencing operator to $m \setminus E$ and $m' \setminus E$. Otherwise, the semantic differencing operator is not applicable to the models and the abstraction. Developers can use abstraction techniques to identify a required abstraction for refinement:

Definition 5 *A required abstraction over a set $A \subseteq \mathcal{M}$ such that a model $m \in M$ refines a model $m' \in M$ is a smallest set $E \subseteq A$ satisfying $m \setminus E \in M$, $m' \setminus E \in M$, and $sem(m \setminus E) \subseteq sem(m' \setminus E)$.*

In Def. 5, the set A contains the model elements to abstract from. The model elements contained in a required abstraction are considered responsible for the existence of semantic differences from m to m' because abstracting from the model elements yields models where the semantic difference is empty. Thus, abstraction techniques detect the syntactic model elements responsible for the existence of semantic differences. Usually, required abstractions are not unique and not guaranteed to exist because not every subset of $\mathcal{M}$ must be a well-formed model. If a required abstraction exists, then it is finite and a subset of the finite set $m \cup m'$ because $m \setminus (m \cup m') = \emptyset = m' \setminus (m \cup m')$.

Algorithm 2 can be used for computing a required abstraction over $A \subseteq \mathcal{M}$ such that a model $m \in M$ refines a model $m' \in M$. The algorithm takes the set of model elements A and the two models m, m' as input. It outputs a required abstraction over A such that m refines m' if such a required abstraction exists. Otherwise, the algorithm returns the special value *nil*. Elementary operations of the algorithm test whether a set of model elements is contained in the set A and check whether a model is a refinement of another model. The algorithm iterates over all sets of modeling elements E contained in m or m' in increasing size (ll. 1-5). If E is a subset of A and abstracting from E in m and m' yields models such that $m \setminus E$ is a refinement of $m' \setminus E$ (l. 2), then the algorithm returns the set E (l. 3). As the algorithm iterates over the sets containing modeling elements of m and m' in increasing sizes, the first set E satisfying the condition is a required abstraction. If no subset of the modeling elements satisfies the condition above, then no required abstraction over A exists. In this case, the algorithm returns the special value *nil* (l. 6).

Algorithm 2 Computing a required abstraction over $A \subseteq \mathcal{M}$ such that a model $m \in M$ refines a model $m' \in M$.

Input: A set of model elements $A \subseteq \mathcal{M}$ and two models $m, m' \in M$.
Output: A required abstraction over A such that m refines m' if such a required abstraction exists. Otherwise, *nil*.

```
1: for all E ⊆ m ∪ m' in increasing sizes do
2:     if E ⊆ A ∧ sem(m \ E) ⊆ sem(m' \ E) then
3:         return E
4:     end if
5: end for
6: return nil
```

2.1.5 Change Sequences for Repairing Refinement

Approaches that combine syntactic and semantic differencing aim at supporting developers in detecting the syntactic model elements that are responsible for the existence of semantic differences [MR15, MR18, KR18a].

Developers can use semantic differencing operators to detect whether a model is a refinement of another model. If the one model is no refinement of the other model, semantic differencing operators usually provide diff witnesses or another model summarizing the semantic difference [FLW11]. Analyzing diff witnesses to identify the syntactic model elements of the one model that cause the existence of semantic differences can be automated by computing shortest repairing sequences [KR18a]. Intuitively, a change sequence repairs a model towards refining another model if it is guaranteed to transform the one model to a refinement of the other model. A change sequence is a shortest change sequence that repairs a model towards refining another model if it repairs the model towards refining the other model and there does not exist any shorter change sequence that repairs the model towards refining the other model.

Definition 6 *A change sequence $t \in O^*$ repairs a model $m \in M$ towards refining a model $m' \in M$ iff $m \rhd t \in M$ and $\emptyset \neq sem(m \rhd t) \subseteq sem(m')$.*

A change sequence $t \in O^*$ is a shortest change sequence that repairs a model $m \in M$ towards refining a model $m' \in M$ iff t repairs m towards refining m' and $\forall u \in O^* : (m \rhd u \in M \wedge \emptyset \neq sem(m \rhd u) \subseteq sem(m')) \Rightarrow |t| \leq |u|$.

The syntactic model elements affected by a shortest repairing change sequence can be interpreted to cause the existence of the semantic differences. The approach presented in [KR18a] for computing shortest repairing change sequences relies on partitioning the change operations applicable to models into finitely many equivalence classes. Each equivalence class contains change operations that *induce an equally quick repair* of the models obtained from applying the operations [KR18a]. Intuitively, two change operations o, p induce an equally quick repair of a model m iff each shortest change sequence that repairs $o(m)$ has the same length as each shortest change sequence that repairs $p(m)$. In general, partitioning the change operations into equivalence classes is not trivial as the set of models M is typically infinite and usually infinitely many change operations are applicable to each model.

2.2 Instantiations of the Framework

This section presents four instantiations of the framework presented in Section 2.1 for existing modeling languages with semantic differencing operators. The CD modeling language and a semantic differencing operator have been introduced in [MRR11d, KMRR17]. The AD modeling language and a semantic differencing operator have been presented in [KR18b]. The SC modeling language and a semantic differencing operator have been introduced in [DEKR19]. The FD modeling language and a semantic differencing operator have been presented in [DKMR19].

2.2.1 Class Diagram Instantiation

The set of models of the CD modeling language [MRR11d, KMRR17] is the set of all well-formed CDs by the definition of well-formedness given in [KMRR17]. The semantic domain is the set of all possible object structures. The semantic mapping assigns a (possibly infinite) set of object structures to every CD. This set represents the possible data states of the system prescribed by the CD.

The semantic differencing operator takes two CDs as input and returns a finite set of object structures that are instances of the first CD and not of the second CD. The implementation is based on a reduction to Alloy[1] [Jac06] as introduced in [MRR11c, KMRR17].

For the summarization of witnesses (see Section 2.1.3), two object structures are considered equivalent if their objects instantiate the same classes [MRR11f]. The abstraction technique considered for CDs allows to abstract from the existence of attributes, classes, enumerations, associations, generalizations, and enumeration fields. The combination of syntactic and semantic differencing as introduced in Section 2.1.5, requires to define a set of syntactic change operations. Similar to [MR15, MR18], the framework-instantiation for CDs therefore offers addition, deletion, and modification of classes, associations, and enumerations. Only existing classes, associations, and enumerations may be modified or deleted and for repairing sequences, we assume that there are finitely many different cardinalities. When computing shortest repairing change sequences, change operations adding classes (roles name or enumeration fields, respectively) not existing in the input CDs are considered to induce an equally quick repair of the one input CD towards refining the other CD.

For example, Figure 2.1 depicts the two CDs `cd.v1` and `cd.v2`. The change sequence $chngCard_{\texttt{Employee},\texttt{Task},\texttt{0..2}}, AddGen_{\texttt{Manager},\texttt{Employee}}, addEnumField_{\texttt{PositionKind},\texttt{external}}$ is a syntactic difference from `cd.v1` to the `cd.v2`. The first operation in the sequence changes the cardinality of the association between the classes `Employee` and `Task` on the end of the class `Task` to `0..2`. The second change operation adds the generalization relation between the classes `Manager` and `Employee`. The third change operation adds the field `external` to the enumeration `PositionKind`.

Applying the semantic differencing operator to compute diff witnesses for the semantic difference from `cd.v2` to `cd.v1` yields, among others, the six diff witnesses presented in Figure 2.2. The object structures `os1` and `os5` are no elements of the semantics of `cd.v1` because there does not exist an association from `Manager` to `Task` in `cd.v1`. The object structures `os2` and `os3` are no elements of the semantics of `cd.v1` because

[1] `http://alloy.mit.edu/`

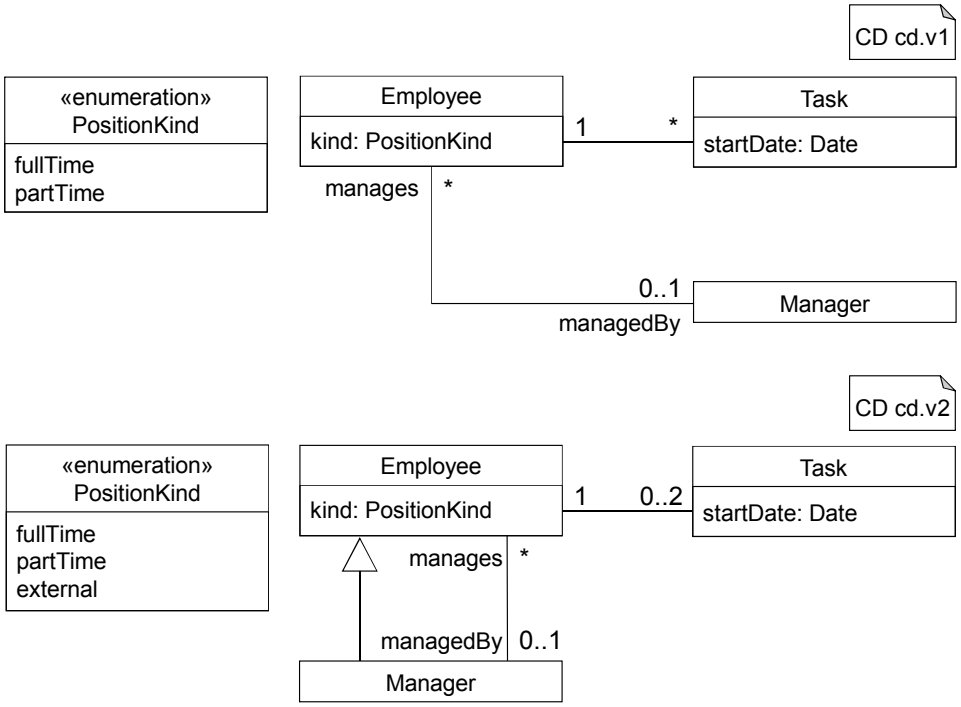


Figure 2.1: Two simple CDs used as examples in the survey that are taken from [MRR11c].

the enumeration field `external` of the enumeration `Position` is not defined in `cd.v1`. The object structures `os4` and `os6` are no elements of the semantics of `cdc.v1` because `Manager` objects can only be linked to `Employee` objects via the role `managedBy` in the CD `cd.v1`.

Considering two object structures that instantiate the same classes as equivalent yields the set $\{\texttt{os2}, \texttt{os5}\}$ as a summary of the set of witnesses $\{\texttt{os1}, \texttt{os2}, \texttt{os5}\}$.

The set $\{\texttt{Gen(Manager, Employee)}, \texttt{EnumField(PositionKind,external)}\}$ is a required abstraction over the set of model elements used in the CDs such that `cd.v2` refines `cd.v1`, by Def. 5. The abstraction applies to the generalization relation between the classes `Manger` and `Employee` and the enumeration field `external` of `PositionKind`.

Similarly, the change sequence containing two change operations that delete the generalization relation between the classes `Manager` and `Employee` and the enumeration field `external` of `PositionKind` is a shortest change sequence for repairing `cd.v2` towards refining `cd.v1`.

2.2.2 Activity Diagram Instantiation

The set of models of the AD modeling language is the set of all well-formed ADs [KR18b]. The semantic domain is the set of all possible finite execution traces over actions. The semantic mapping assigns a (possibly infinite) set of execution traces to each AD, containing only actions that are explicitly modeled in the AD.

The semantic differencing operator takes two ADs as input and returns a finite set of execution traces that are modeled in the one AD not modeled in the other AD. In [KR18a],

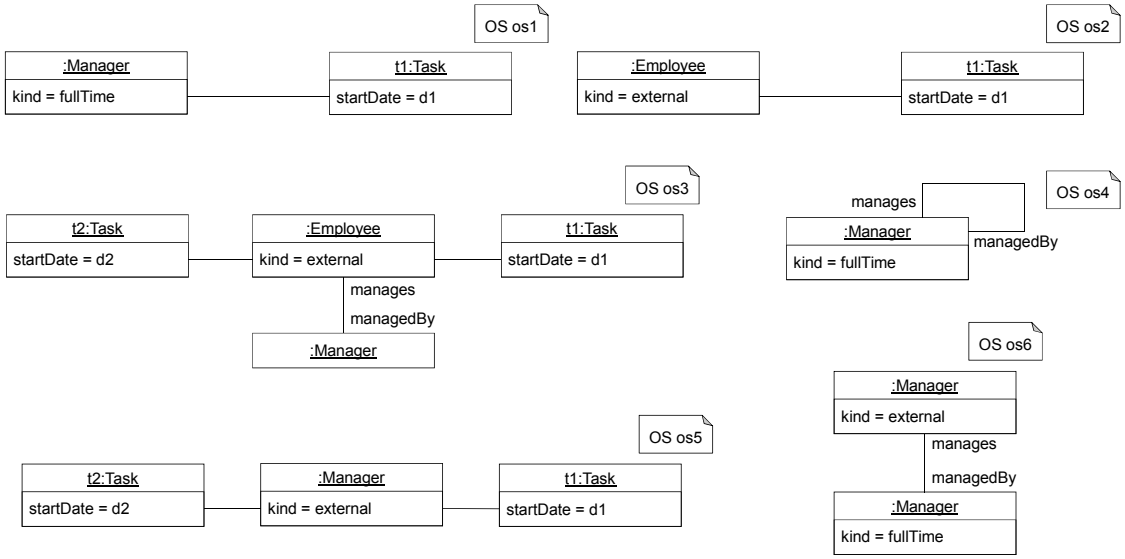


Figure 2.2: Six object structures contained in the semantic difference from the CD `cd.v1` to the CD `cd.v2` (cf. Figure 2.1).

the semantics of ADs is based on a translation to finite automata [HMU06], which reduces semantic differencing of ADs to language inclusion checking between finite automata. The implementation of the semantic differencing operator uses the automaton language inclusion checking tool RABIT[2].

For the summarization of witnesses, two execution traces are considered equivalent if the sets of actions appearing in the traces are equal to each other [MRR11f]. For the study, we use an abstraction technique that abstracts from the existence of actions, xor-fragments, and-fragments, cyclic-fragments, and transitions. The abstraction from actions or fragments connects the control flow from the action's or fragment's predecessor node to the respective successor node accordingly if it is not abstracted from the transitions. We use syntactic change operations inspired by [MR15, MR18, KR18a], *i.e.,* insertion, deletion of actions and fragments as well as type-conversion for fragments. The partitioning of the change operations and the algorithm for the computation of repairing sequences are described in [KR18a].

For example, Figure 2.3 depicts the two ADs `ad.v1` and `ad.v2`. An example of a syntactic difference from `ad.v1` to `ad.v2` is the change sequence that contains change operations to execute the following changes:

1. delete the action labeled `Check Claim`,
2. insert a cyclic-fragment between the nodes `Record Claim` and the following decision node,
3. add an action labeled `Check Claim` to the cyclic-fragment between the merge and the decision node,
4. add an action labeled `Retrieve Add. Data` to the cyclic fragment between the decision node and the merge node,

[2] `http://www.languageinclusion.org/`

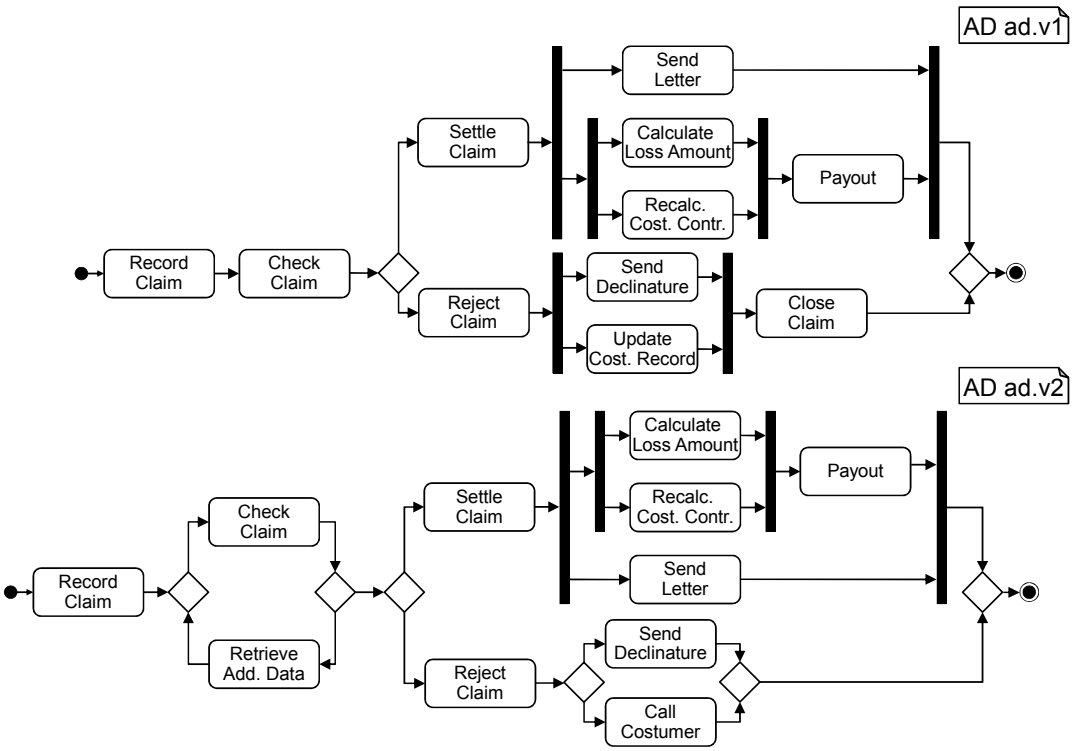


Figure 2.3: Two simple ADs used as examples in the survey that are inspired by similar ADs from [MR15, MR18, KR18a].

5. convert the and-fragment containing the actions labeled `Send Declinature` and `Update Cost. Record` to an xor-fragment,
6. delete the action labeled `Update Cost. Record`,
7. add the action labeled `Call Costumer` to the xor-fragment containing the action labeled `Send Declinature`, and
8. delete the action labeled `Close Claim`.

Applying the semantic differencing operator to compute diff witnesses for the semantic difference from `ad.v2` to `ad.v1` yields, among others, the five diff witnesses presented in Figure 2.4. Each of the execution traces is modeled in `ad.v2` and not modeled in `ad.v1`.

With the summarization equivalence relation on the set of execution traces as described above, summarizing execution traces whose sets of actions are equal yields the set of witnesses containing exactly the first, the second, the fourth, and the fifth execution traces depicted in Figure 2.4. The first execution trace is equivalent to the third execution trace. The other execution traces are pairwise not equivalent.

The set containing the syntactic elements for the nodes labeled `Retrieve Add. Data`, `Update Cost. Record`, `Call Costumer`, and `Close Claim` (and the transitions of the nodes labeled `Retrieve Add. Data`, `Call Costumer`, and `Close Claim`) is a required abstraction over the set of model elements used in the ADs such that `ad.v2` refines `ad.v1`.

A shortest change sequence that repairs `ad.v2` towards refining `ad.v1` is one that contains the change operations that prescribe the following transformations:

```
1. Record Claim, Check Claim, Retrieve Add. Data, Check Claim, Settle Claim,
   Send Letter, Calculate Loss Amount, Recalc. Cost. Contr., Payout

2. Record Claim, Check Claim, Retrieve Add. Data, Check Claim, Reject Claim,
   Send Declinature

3. Record Claim, Check Claim, Retrieve Add. Data, Check Claim, Settle Claim,
   Calculate Loss Amount, Recalc. Cost. Contr., Send Letter, Payout

4. Record Claim, Check Claim, Reject Claim, Call Costumer

5. Record Claim, Check Claim, Retrieve Add. Data, Check Claim, Reject Claim,
   Call Costumer
```

Figure 2.4: Five diff witnesses contained in the semantic difference from `ad.v2` to `ad.v1`.

1. delete the action node labeled `Retrieve Add. Data`,
2. delete the action node labeled `Call Costumer`,
3. add an action node labeled `Close Claim` after the action node labeled `Send Declinature` and reconnect the control flow accordingly,
4. add an action node labeled `Update Cost. Record` after the action node labeled `Send Declinature` and reconnect the control flow accordingly.

2.2.3 Statechart Instantiation

The set of models of the SC modeling language [DEKR19] is the set of all well-formed SCs by the definition of well-formedness given in [DEKR19]. The semantic domain is the set of all possible finite stimulus/reaction traces. The semantic mapping maps each SC to the (usually infinite) set of all stimulus/reaction traces that it describes.

The semantic differencing operator takes two SCs as input and outputs a set of stimulus/reaction traces that are possible in the one SC and not in the other SC. Semantic differencing of SCs can be reduced to language inclusion checking for finite automata [DEKR19, HMU06]. This enables an implementation based on RABIT.

For the summarization of witnesses, two traces are considered equivalent if the sets of stimulus/reaction pairs used in the traces are equal. We use an abstraction technique that allows to abstract from the existence of events in the reactions of transitions [DEKR19]. For combining syntactic and semantic differencing, we consider change operations for adding and deleting states and transitions and changing the initial state. The partitioning of the change operations is similar to the partitioning of the change operations for the time-synchronous port automaton [BKRW17, BKRW19] modeling language described in [KR18a]. Operations for adding states that add states not used in the SCs are pairwise equivalent and change operations for adding possible input and output events that add events not used in the SCs are pairwise equivalent.

For example, Figure 2.5 depicts the two SCs `sc.v1` and `sc.v2`. A syntactic difference from `sc.v1` to `sc.v2` is, for example, given by a change sequence that contains two change operations that prescribe the following transformations:

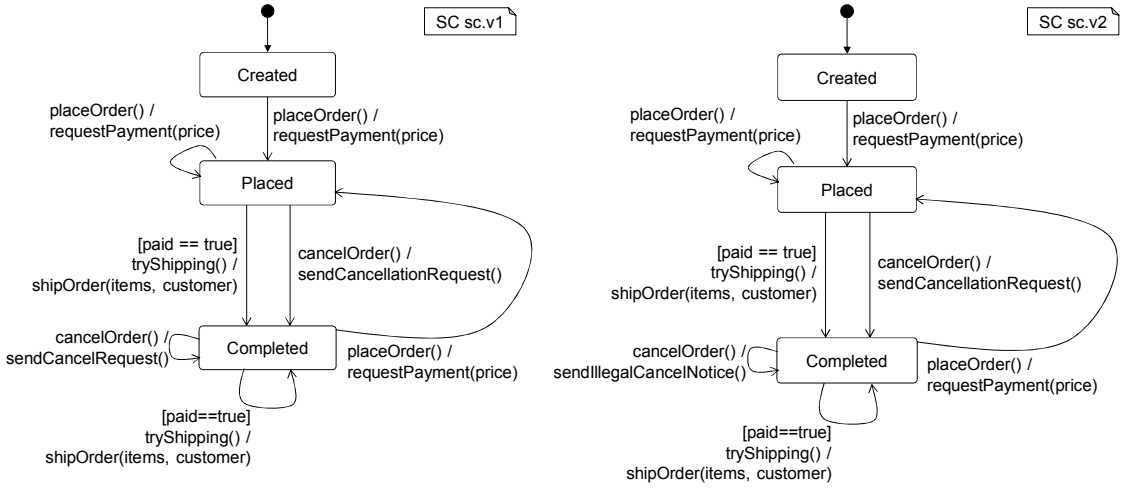


Figure 2.5: Two simple SCs used as examples in the survey. The SCs are taken from [DEKR19].

1. add a transition from the state `Completed` to the state `Completed` with the stimulus `cancelOrder()` and the reaction `sendIllegalCancelNotice()`.

2. delete the transition from the state `Completed` to the state `Completed` with the stimulus `cancelOrder()` and the reaction `sendCancelRequest()`.

Applying the semantic differencing operator to compute diff witnesses for the semantic difference from `sc.v2` to `sc.v1` yields, among others, the diff witnesses t_1, t_2, t_3 presented in Figure 2.6. The traces t_1, t_2 and t_3 are no traces in the semantics of `sc.v1` because there are no executions of `sc.v1` that produce the traces. The bottom of Figure 2.6 depicts the execution e_1 of `sc.v2` producing the trace t_1.

With the summarization equivalence relation on the set of stimulus/reaction traces as described above, summarizing the traces with equal stimulus/reaction pairs, yields the set of witnesses $\{t_1, t_2\}$. The traces t_2 and t_3 are equivalent.

The set containing the syntactic model elements causing the existence of the actions `sendIllegalCancelNotice()` and `sendCancelRequest()` in the reactions of the transitions of the SCs looping in the state `Completed` with the stimulus `cancelOrder()` is an abstraction over the set of actions used in the reactions of the transitions of the SCs such that `sc.v2` refines `sc.v1`. If the actions `sendIllegalCancelNotice()` and `sendCancelRequest()` did not exist in the reactions of the transitions of the SCs, then the SC `sc.v2` would be a refinement of the SC `sc.v1`. It is also possible to apply event matching for detecting the syntactic SC model elements causing that the one SC is no refinement of the other SC [DEKR19]. For the example SCs depicted in Figure 2.5, replacing all occurrences of the event `sendIllegalCancelNotice()` in the SCs with the event `sendCancelRequest()` yields two modified SCs such that the semantic difference from the modification of `sc.v2` to the modification of `sc.v1` is empty.

The change sequence containing a change operation for deleting the transition (`Completed`, `cancelOrder()/sendIllegalCancelNotice(),Completed`) is a shortest change sequence that repairs `sc.v2` towards refining `sc.v1`.

```
t1 = placeOrder() / requestPayment(price),
     tryShipping() / shipOrder(items, customer),
     placeOrder() / requestPayment(price),
     cancelOrder() / sendCancellationRequest(),
     cancelOrder() / SendIllegalCancelNotice(),
     cancelOrder() / SendIllegalCancelNotice()

t2 = placeOrder() / requestPayment(price),
     tryShipping() / shipOrder(items, customer),
     cancelOrder() / SendIllegalCancelNotice()

t3 = placeOrder() / requestPayment(price),
     placeOrder() / requestPayment(price),
     tryShipping() / shipOrder(items, customer),
     cancelOrder() / SendIllegalCancelNotice(),
     tryShipping() / shipOrder(items, customer)
```

```
e1 = (Created, placeOrder() / requestPayment(price), Placed),
     (Placed, tryShipping() / shipOrder(items, customer), Completed),
     (Completed, placeOrder() / requestPayment(price), Placed),
     (Placed, cancelOrder() / sendCancellationRequest(), Completed),
     (Completed, cancelOrder() / SendIllegalCancelNotice(), Completed),
     (Completed, cancelOrder() / SendIllegalCancelNotice(), Completed)
```

Figure 2.6: Three diff witnesses t_1, t_2, t_3 contained in the semantic difference from `sc.v2` to `sc.v1` and an execution e_1 of `sc.v2` producing the trace t_1.

2.2.4 Feature Diagram Instantiation

The set of models of the FD modeling language [DKMR19] is the set of all well-formed FDs. The semantic domain is the set of all possible configurations (finite sets of features). The semantic mapping maps each FD to the set of all configurations that are valid in the FD and solely contain features used in the FD. The semantic mapping represents the closed-world semantics in [DKMR19]. The closed-world semantics requires configurations of an FD to solely contain features of the FD. In contrast, the open-world semantics [DKMR19] allows configurations of an FD to contain features that are not contained in the FD, as long as the configuration satisfies all constraints imposed by the FD.

The semantic differencing operator takes two FDs as input and returns a finite set of configurations that are valid in the one FD not valid in the other FD. It is possible to provide an implementation based on the Boolean satisfiability problem for propositional logic [DKMR19]. This enables, for instance, an implementation based on Alloy [Jac06].

For the summarization of witnesses, two configurations are considered equivalent if they have the same cardinality. We use an abstraction technique that enables to abstract from the existence of subtrees of the FDs and from cross-tree constraints. The syntactic change operations we consider for syntactic differencing and for combining syntactic with semantic differencing are inspired by [MR15, MR18, KR18a]. We use change operations for creating and deleting features without children and cross tree constraints, changing the types of groups, making optional features mandatory, making mandatory features optional, and creating groups with features. The partitioning of the change operations is similar to the partitioning described in [KR18a].

For example, Figure 2.7 depicts the two FDs `fd.v1` and `fd.v2`. A syntactic difference from `fd.v1` to `fd.v2` is, *e.g.,* a change sequence that contains change operations that prescribe the following transformations:

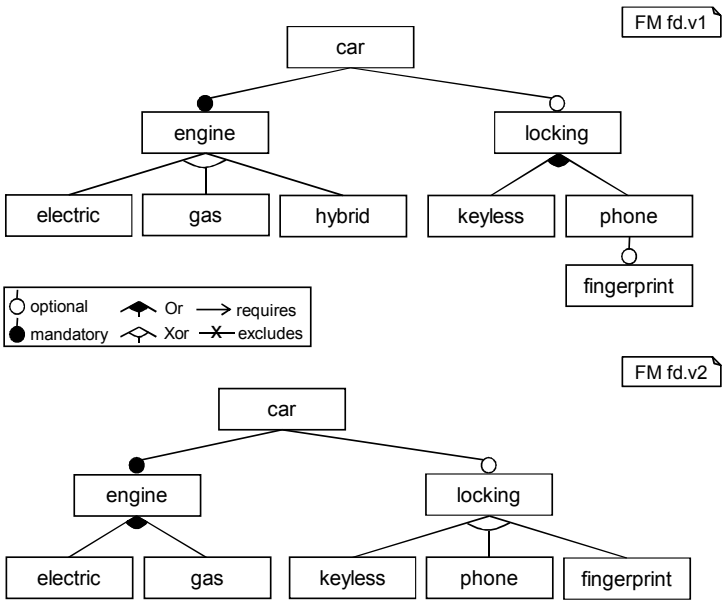


Figure 2.7: Two simple FDs used as examples in the survey. The FDs are taken from [MR15, MR18, KR18a].

C_1 ={car, engine, electric, gas, locking, keyless}
C_2 ={car, engine, electric, gas, locking, phone}
C_3 ={car, engine, electric, gas}
C_4 ={car, engine, electric, gas, locking, fingerprint}
C_5 ={car, engine, gas, locking, fingerprint}

Figure 2.8: Five diff witnesses contained in the semantic difference from `fd.v2` to `fd.v1`.

1. convert the group with the parent feature `engine` containing the features `electric`, `gas`, `hybrid` to an or-group,

2. delete the feature `hybrid` from the group with the parent feature `engine` containing the features `electric`, `gas`, `hybrid`,

3. delete the feature `fingerprint`,

4. convert the group with the parent feature `locking` containing the features `keyless` and `phone` to an xor-group,

5. add the feature `fingerprint` to the group with the parent feature `locking` containing the features `keyless` and `phone`.

Applying the semantic differencing operator to compute diff witnesses for the semantic difference from `fd.v2` to `fd.v1` yields, among others, the diff witnesses presented in Figure 2.8. The configurations C_1, C_2, C_3, and C_4 are not valid in `fd.v1` , *e.g.,* because the FD models that the features `electric` and `gas` must not be chosen simultaneously. The configuration C_5 is not valid in `fd.v1` because the FD requires to also choose the feature `phone` if the feature `fingerprint` is chosen.

By considering the configurations of equal cardinality as equivalent, a summary of the set of witnesses is the set containing the configurations C_3, C_4, and C_5 depicted in Figure 2.4.

The set containing the syntactic model elements causing the existence of the features `fingerprint` and `gas` is a required abstraction over the set of model elements used in the FDs such that `fd.v2` refines `fd.v1`. If the features `fingerprint` and `gas` did not exist in the FDs, then the FD `fd.v2` would be a refinement of the FD `fd.v1`. Similarly, the set containing the syntactic model elements causing the existence of the features `fingerprint` and `electric` is also a required abstraction over the set of model elements used in the FDs such that `fd.v2` refines `fd.v1`.

The change sequence containing two change operations for performing the following changes is a shortest change sequence that repairs `fd.v2` towards refining `fd.v1`:

1. delete the feature `fingerprint`,
2. convert the group with the parent feature `engine` containing the features `electric` and `gas` to an xor-group.

Chapter 3

Study Design

We have created a questionnaire including several examples for the syntactic and semantic differencing of models of the four modeling languages under investigation (cf. Appendix A). We did a pre-test and adjusted the questionnaires' examples for better understandability.

Hypotheses. Based on our research results, we state the following hypothesis including its sub-parts: Using semantic differencing improves the understandability between differences of model versions.

H1 Using syntactic differencing improves the understandability between differences of model versions.

H2 Using semantic differencing helps more than syntactic differencing.

H3 Using syntactic and semantic differencing in combination helps more than syntactic and/or semantic differencing alone.

H4 Using the abstracted version of semantic differencing helps more than syntactic differencing and/or the full version of semantic differencing.

H5 Using the summarized version of semantic differencing helps more than syntactic differencing and/or the full version of semantic differencing.

H6 There exists a variety of use cases where the difference between models could be used by software engineers in practice.

These aspects together constitute the hypothesis and, therefore, are used for the evaluation of the hypothesis. The questions for the modeling languages under investigation are based on these aspects.

Structure of the Questionnaire. The questionnaire included six parts: (A) Introduction questions, (B) Class Diagrams, (C) Activity Diagrams, (D) Statecharts, (E) Feature Diagrams and (F) final remarks. The questionnaire language was German, as all participants were speaking German. For a better understanding, we have translated the main aspects of the questionnaire to English for the descriptions in this report.

Part (A) included a question about the expertise of the participants regarding each of the four investigated modeling languages on a four aspect scale (very good, good, low, very

low) as well as one question about how long they are modeling (0 years, >0 to <=5 years, >5 to <=10 years, >10 years).

Parts (B) - (E) had the same structure and set of questions for all four modeling languages:

1. Introductory text
2. Text explaining the use case and two model versions
3. Textual explanation of the syntactic differences
4. Graphical explanation of the syntactic differences
5. Textual explanation providing witnesses of the semantic differences (not for CD)
6. Graphical examples of the semantic diff witnesses
7. Textual explanation of the combination between syntactic and semantic differences
8. Explanation of the semantic differences using an example for abstraction
9. Explanation of the semantic differences using summarization
10. Five questions about these examples

The questionnaire starts with an introductory text about the differences between two models in this modeling language, whereas the model labeled with v2 is the successor of the one labeled with v1. In Figure 2.3, section 2.2.2 such an example for ADs was already shown. These models are accompanied by a short explanation about the specific use case.

Afterwards, the syntactic difference between these models is explained in textual as well as in graphical form. Figure 3.1 shows the graphical representation of the differences for the ADs. Elements marked in red and surrounded by a box in v1 were deleted or changed, elements marked in green and surrounded by a box in v2 were added or changed.

Then, the semantic difference from model v2 to model v1 is explained in textual as well as in graphical form. The textual form shows examples/instances which are possible in v2 but not possible in v1, e.g, for ADs traces which are possible in v2 but not in v1. Figure 3.2 and Figure 3.3 depict execution traces for the ADs in a graphical form, as also presented in the questionnaire.

The next part shows the combination of syntactic and semantic differences by providing a step-by-step instruction of changes (model repair) to be made on v2 to obtain a model that is a refinement of v1.

For an explanation of the abstraction of semantic differences only the main elements of the modeling languages are considered. The individual modeling elements for the specific modeling languages are described in Section 2.2.

Using summarization is the last mentioned aspect to investigate the semantic differences between models. Thus, the elements of a subset of the examples/instances shown in the textual description of the semantic differences were checked for equivalence and only one of the equivalent examples/instances was presented. e.g., for ADs only three out of five execution traces were presented.

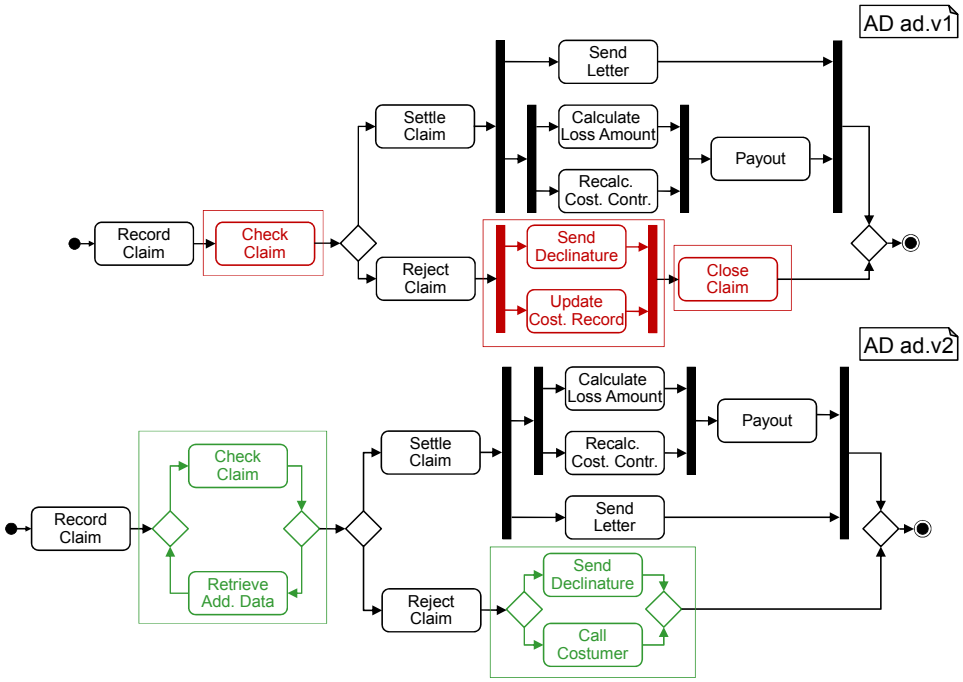


Figure 3.1: The syntactic differences marked within the two versions of the AD.

The last page of parts (B) - (E) included a set of questions which were related to these examples and had to be answered using a four point scale (very good, good, little, very little). We were asking how well the following aspects helped to understand the differences between the models:

Q B-E.1: Syntactic differences

Q B-E.2: Semantic differences

Q B-E.3: Combination of syntactic and semantic differences

Q B-E.4: Abstraction of semantic differences

Q B-E.5: Summarization of semantic differences

As the questionnaire did already include many questions, we did not further differentiate on the textual and graphical representation of the syntactic and semantic differences within the questions but used both representations.

Part (F) included one open question about use cases where such syntactic and semantic differencing operators could help software developers.

Pre-Test. In a pre-test the examples and questions were shown to two software engineers. Using their feedback, we have slightly revised the examples for better understandability.

Study Participants. Participants in the study were 18 software engineers and students from RWTH Aachen University with mixed experiences using models in the four investigated languages. As the number of participants is too small to make further differences regarding their modeling experiences in the languages as well as in the duration how long they are modeling, we did not split the group further in our evaluation.

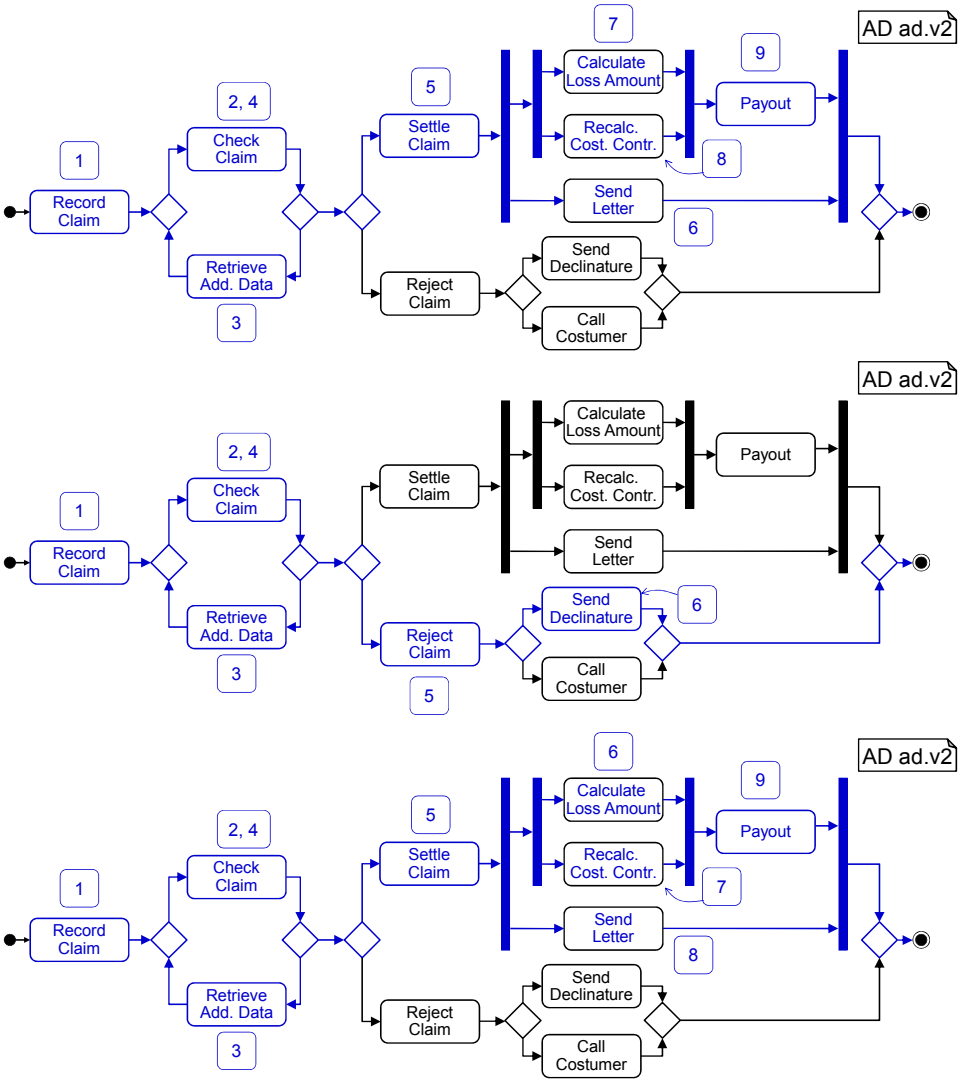


Figure 3.2: Diff witnesses in the semantic difference from `ad.v2` to `ad.v1` highlighted in the AD `ad.v2`.

Setting. The participants received the questionnaire on paper (34 pages, see Appendix A) and had two days to answer the questions.

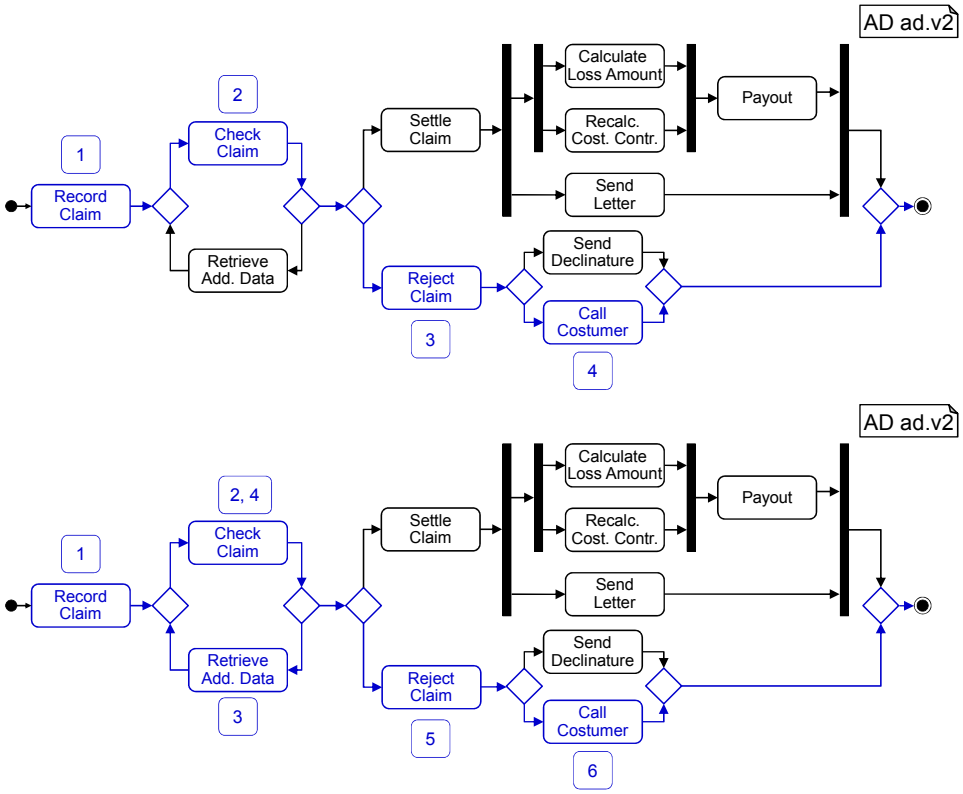


Figure 3.3: Diff witnesses in the semantic difference from `ad.v2` to `ad.v1` highlighted in the AD `ad.v2`.

Chapter 4

Study Results

This section presents the results of the survey in terms of the percentages of how often the answers were chosen by the participants. In total, we received answers from 18 participants. We included the first question regarding the modeling experience to interpret our results with respect to the experiences of the participants. Questions 1-5 of parts B-E aim to confirm or falsify hypotheses H1 to H5 presented in the previous section. Therefore, we draw a short conclusion regarding each hypotheses after presenting the results to the respective questions. To interpret the results, we used the following likert scale:

- very good $\rightarrow$ 1,
- good $\rightarrow$ 2,
- little $\rightarrow$ 3,
- very little $\rightarrow$ 4.

4.1 Modeling Experience

To interpret the results of the modeling experience, we used a likert scale from one to four, mapping the years of experience to numbers as follows:

- 0 years $\rightarrow$ 1,
- $>$ 0 to $<=$ 5 years $\rightarrow$ 2,
- $>$ 5 to $<=$ 10 years $\rightarrow$ 3,
- $>$ 10 $\rightarrow$ 4.

Figure 4.1 shows the results of how experienced our participants were when conducting the survey: Out of all participants, 72% have between zero and five years, 17% have between five and ten years, and 6% have more than ten years of general modeling experience.

Figure 4.2 depicts the results of how experienced the participants were in using the regarded modeling languages. Half of the participants had no experience (0 years) (50%), 6% had more than ten years of experience, and 44% had between zero and five years of experience in modeling with CDs. Of all participants, 17% had no modeling experience with ADs, 72% had zero to five years of experience, 5% had five to ten years, and 6% had more than ten years of experience. For SCs, 17% of the participants had no experience

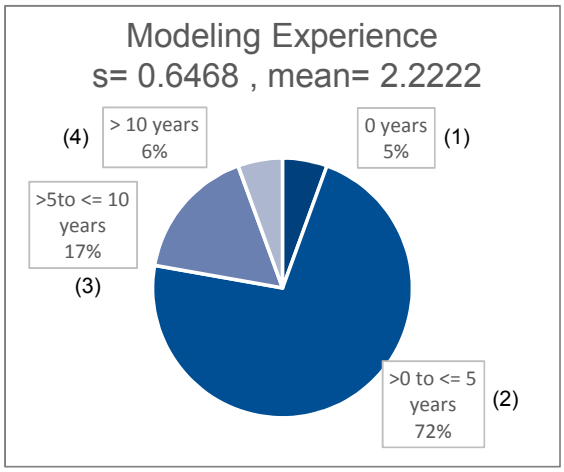


Figure 4.1: General modeling experience among participants (Q A.2).

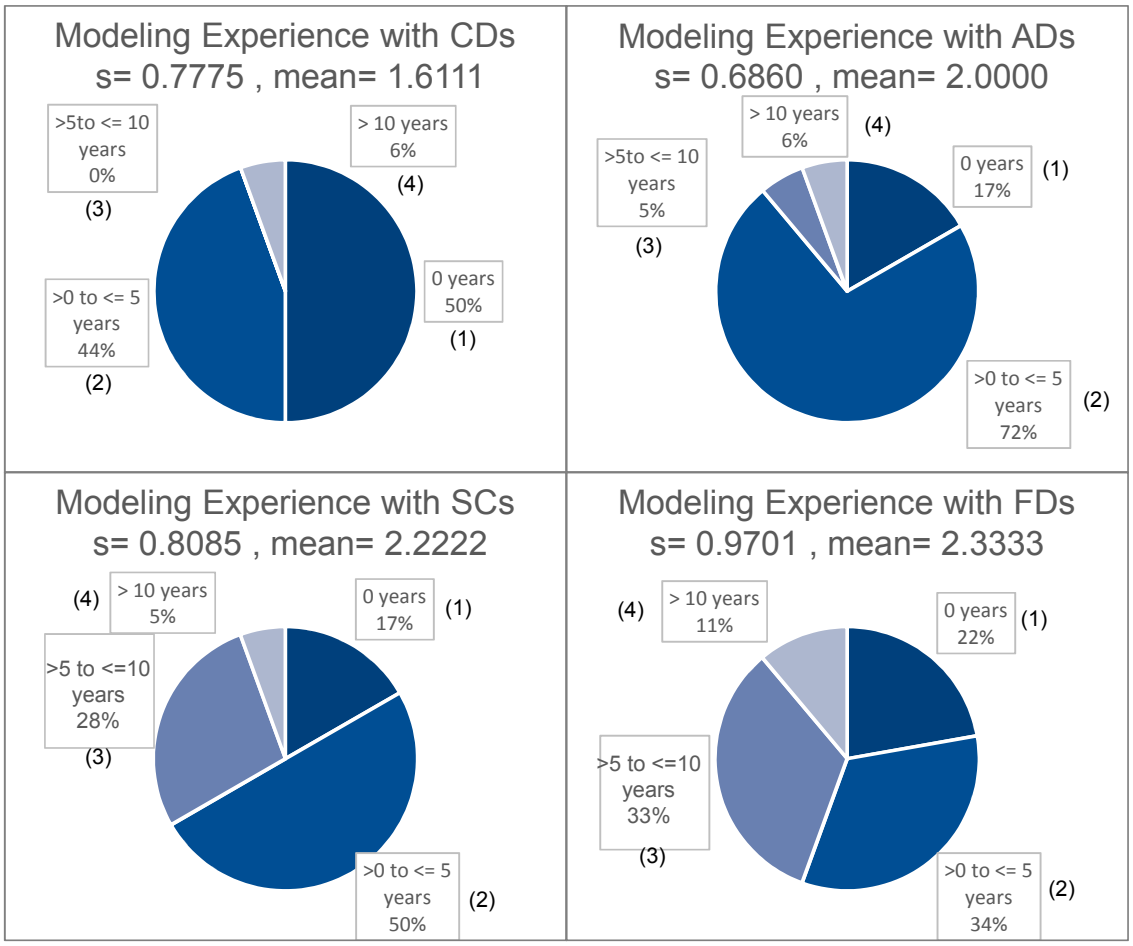


Figure 4.2: General modeling experience among participants (Q A.1).

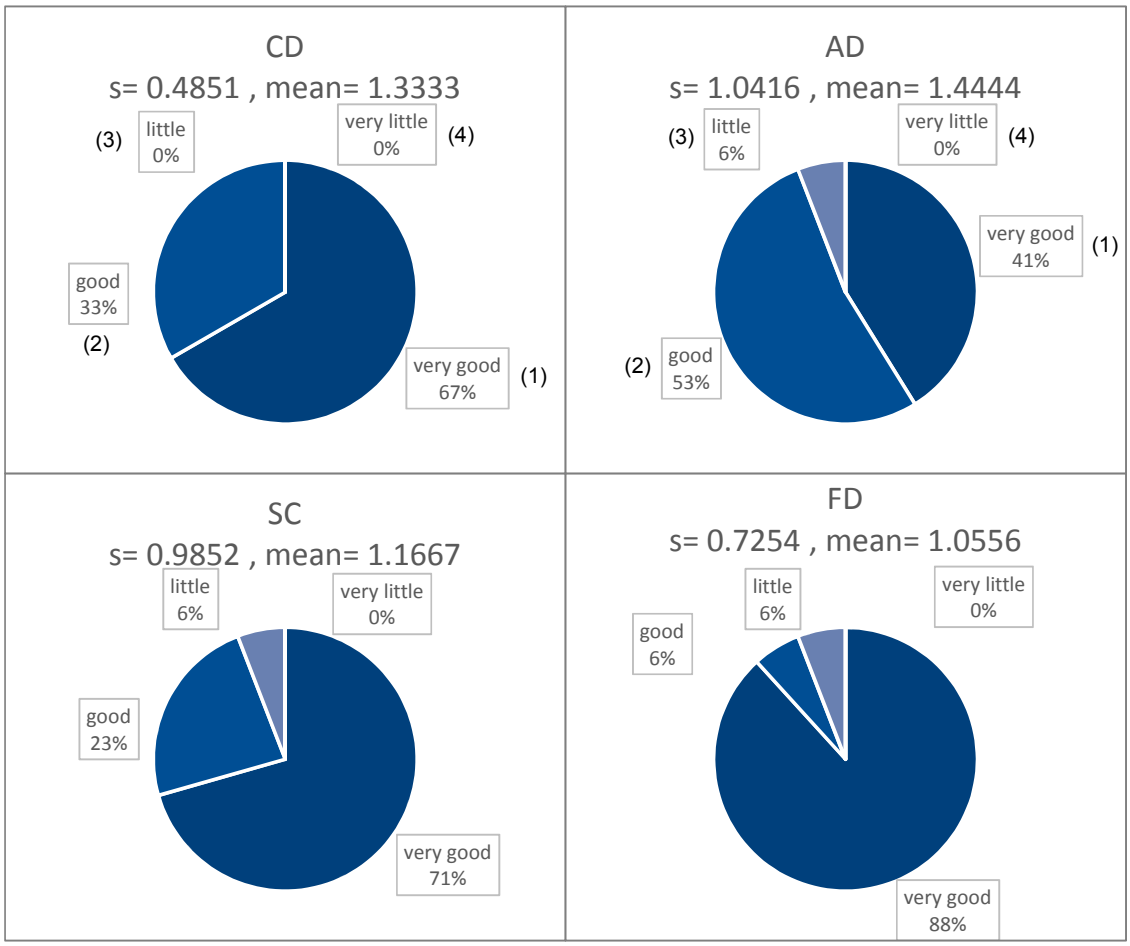


Figure 4.3: Percentage of the answers to Q B.1-E.1 about the helpfulness of syntactic differences for understanding model differences.

with SCs, 50% had between zero and five years of experience, 28% had between five and ten years, and 5% had more than ten years of experience. Of all participants, 22% had no FD-experience, 34% had zero to five years of experience, 33% had between five and ten years of experience, and 11% had more than ten years of experience.

Conclusion According to the question regarding general modeling experience, we conclude that the participants were mostly inexperienced or low-experienced, with zero to at most five years of modeling experience. The low standard deviation indicates also that most of our participants were equally (low) experienced. The results regarding the experience utilizing the targeted modeling languages were anticipated by the results of the general modeling experience. However, for FDs, the spectrum of modeling experience was broader (higher standard deviation) and included also quite experienced users as indicated by the quite high mean value.

4.2 Syntactic Differences

Figure 4.3 shows the results to questions B.1-E.1 about the helpfulness of syntactic differences for understanding model differences for each language. For CDs, 67% found the syntactic difference very helpful (very good) to understand the model differences, while 33% found them to be helpful (good) (cf. diagram at the top left of Figure 4.3). For ADs, 41% of the participants found the syntactic difference to be very helpful, 53% helpful, and 6% answered that the syntactic difference helped little (little) to understand the model differences (cf. diagram at the top right of Figure 4.3). In case of SCs, the syntactic difference was considered very helpful by 71% of the participants, helpful by 23%, and little helpful by 6% of the participants as the diagram at the bottom left of Figure 4.3 shows. Lastly, 88% considered the syntactic difference helpful for understanding differences between the two FDs, while only 6% considered them helpful or little helpful, respectively (cf. diagram at the bottom right of Figure 4.3).

Conclusion: The results hint at hypothesis H1 being correct, since for all considered languages more than 90% of the participants considered the syntactic difference very helpful or helpful to understand the model differences. All mean values are close to one, meaning the syntactic differences are very helpful for understanding the model differences for all modeling languages. For ADs and SCs, the standard deviation is relatively high, however, the answers mostly range between very good and good.

4.3 Semantic Differences

The top left of Figure 4.4 shows that for CDs, 28% of the participants either found the semantic difference very helpful or little helpful, respectively. At the same time, 33% considered them helpful, and for 11% the semantic difference helped only very little to understand the CD differences (very little).

The results are similar for ADs, which Figure 4.4 shows in the diagram at the top right. Again, the same percentage, i.e., 35%, of the participants considered the semantic difference either very helpful or little helpful. At the same time, 18% found the semantic difference helpful and 12% found it to help very little in understanding the AD differences.

Figure 4.4 shows the results for SCs at the bottom left. In this case, only 17% of the participants considered the semantic SC difference to be very helpful and only 11% found it helpful. The semantic difference was considered a little helpful to understand model differences by 44% and very little by 28%.

For FDs, the result resembles that of CDs and ADs: Of the participants, 35% considered the semantic FD difference very helpful, 30% considered it helpful, 29% a little helpful, and 6% very little helpful to understand the FD differences.

Conclusion These results show that the helpfulness of semantic differences to understand model differences is perceived more diversely, which is also reflected by the relatively high standard deviations (approximately one for all languages). Looking at the absolute results for CDs, ADs and FDs, the perception of the participants was controversial, as they considered the semantic difference either very helpful or little helpful. This indicates

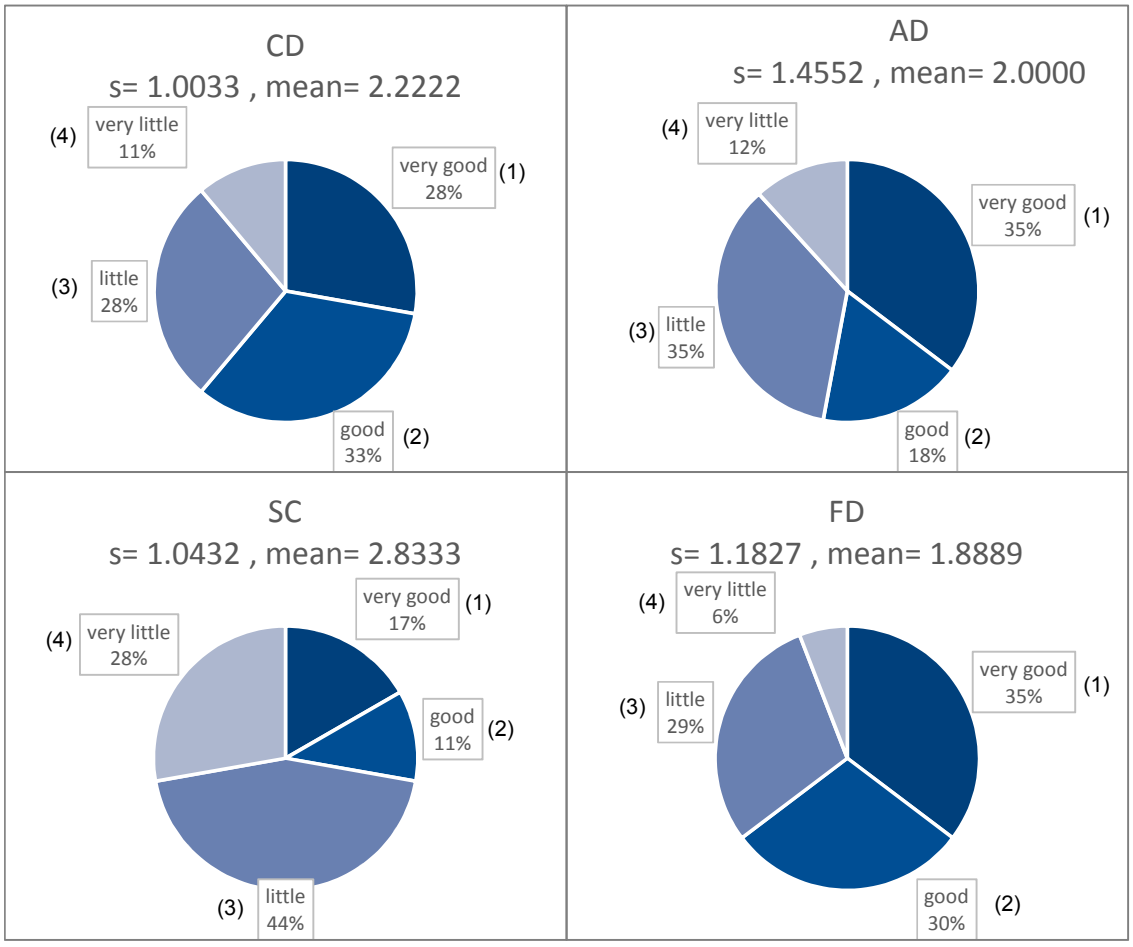


Figure 4.4: Percentages of the answers to Q B.2-E.2 about the helpfulness of semantic differences for understanding model differences.

that the helpfulness of the semantic difference to understand model differences between CDs, ADs, and FDs is subjective or dependent on other aspects than considered in this study. In case of SCs, the semantic difference does not seem to be useful as 72% of the participants found it to only be little or even very little helpful in understanding the differences. The standard deviation is high (1.04), however the high mean of 2.8 shows that the answers vary between little and very little helpful. For SCs and FDs, our participants were most experienced as 28% and 33%, respectively, stated they had 5 to ten years of experience and 50% and 34%, respectively, stated zero to 5 years of experience with SCs and FDs, respectively. Since participants are rather experienced in these languages, we consider the answers for SCs and FDs to be more meaningful.

The results, therefore, pretty much falsify H2, since more than 90% considered the syntactic difference very helpful or helpful. For CDs, ADs, and FDs, more than half of the participants considered the semantic difference very helpful or helpful. For SCs, the result is even more striking, since 72% of all participants found the semantic difference little or very little helpful.

4.4 Combination of Syntactic and Semantic Differences

Figure 4.5 depicts the results of our study. The diagram at the top left of Figure 4.5 shows the results for CDs: 28% of the participants considered the combination to be very helpful to understand the CD differences, 33% considered it to be helpful or little helpful, respectively, and 6% considered it to provide only very little help. For ADs, 22% found the combination to be very helpful for understanding the AD differences, 45% found it helpful, and 33% little helpful. The combined approach was considered very helpful for understanding SC differences by 22%, and helpful by 28%. Half of the participants found it little helpful. For FDs, 29% found the combination very helpful, 47% helpful, 18% considered it little helpful, and 6% very little helpful to understand FD differences.

Conclusion The results indicate that hypothesis H3 is not correct for syntactic differencing, since the syntactic difference was considered very helpful or helpful by a large majority of the participants. When it comes to semantic difference, whether or not the combination is more helpful than the semantic difference alone seems to be language dependent. For CDs and FDs, the mean and standard deviation for the combination differ marginally from those for the semantic difference. For these languages, the results indicate that the combination is roughly equally well suited for explaining model differences as the semantic difference by itself. For ADs, the mean values for combination and semantic difference are very close, however the standard deviation for the combination is significantly lower. The results indicate that, for ADs, the combination is a better explanation for model differences, since none of the participants considered the combination very little helpful and the number of participants who considered the combination at least helpful is higher. Most of the participants considered the semantic difference alone little or very little helpful for understanding SC differences, i.e., 72%. The combination, for SCs, seems to improve the understanding of model differences, which becomes apparent by comparing the mean values. In the majority of the cases, the combination of syntactic and semantic differencing was considered to be at least helpful.

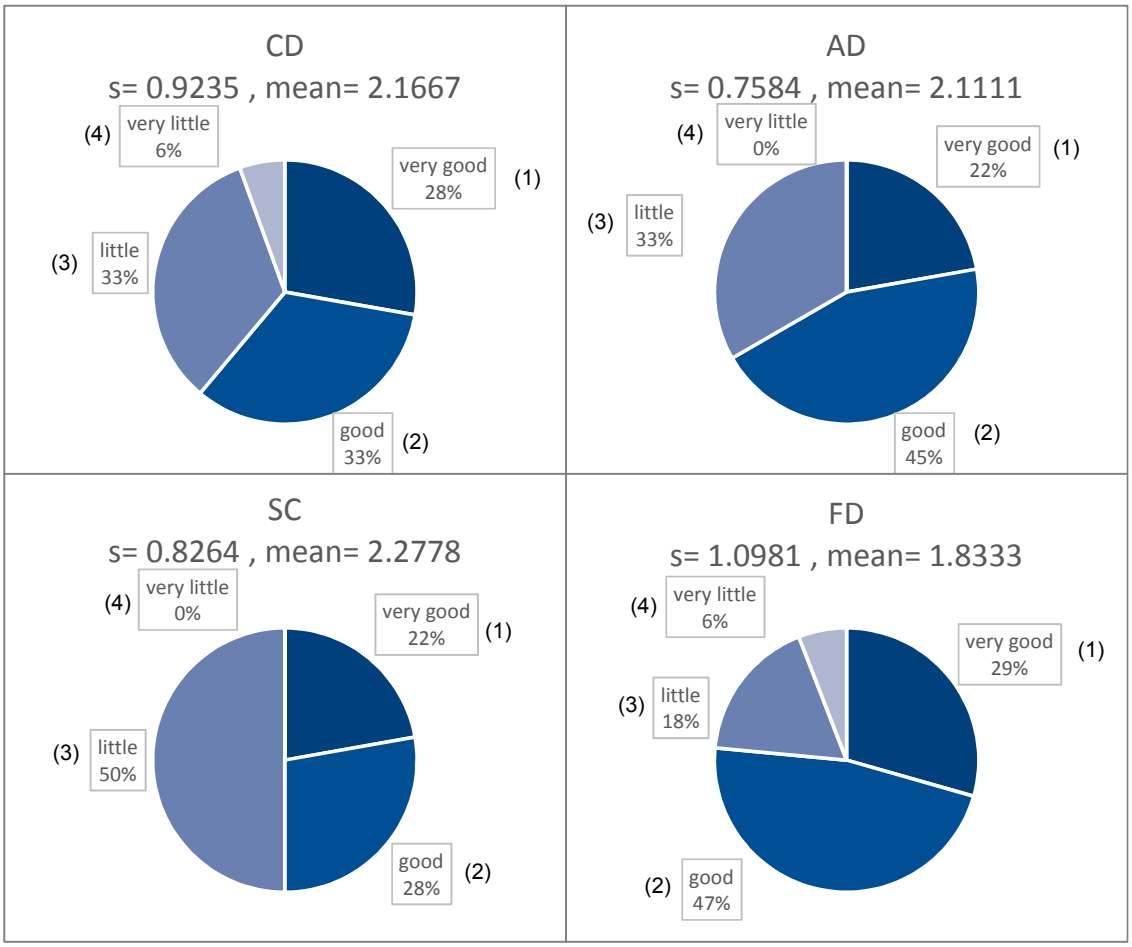


Figure 4.5: Percentages of the answers to Q B.3-E.3 about the helpfulness of the combination of syntactic and semantic differences for understanding model differences.

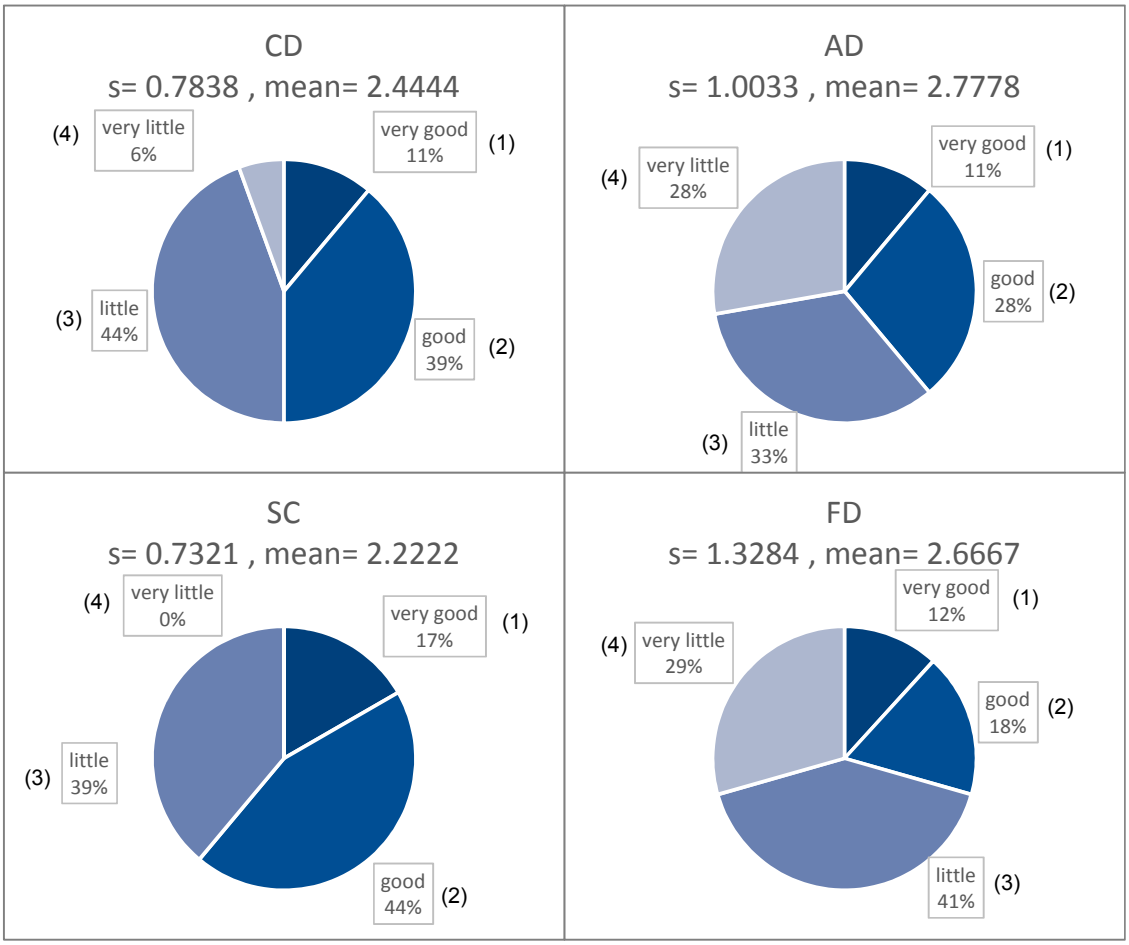


Figure 4.6: Percentages of the answers to Q B.4-E.4 about the helpfulness of abstraction for understanding model differences.

4.5 Abstraction

For CDs, 11% of the participants found the abstraction very helpful, 39% found it helpful, 44% little helpful, and 6% very little helpful to understand the CD differences (cf. top left of Figure 4.6). Of all the participants, 11% found the abstraction to be very helpful for understanding AD differences, 28% found it helpful, 33% found it little helpful, and 28% found it very little helpful (cf. top right of Figure 4.6). In case of SCs, 17% found the abstraction very helpful, 44% found it helpful, and 39% found it little helpful, as the diagram at the bottom left of Figure 4.6 shows. For FDs, 12% of the participants found the abstraction very helpful, 18% found it helpful, 41% found it little helpful, and 29% found it to help very little to understand FD differences (cf. bottom right of Figure 4.6).

Conclusion The results indicate that hypothesis H4 is incorrect regarding the syntactic differences which were considered very helpful or helpful by more than 90% of the participants, while the percentages of the participants considering abstraction as very helpful or helpful range from 30% for FDs to 61% for SCs.

For CDs, ADs, and FDs, the mean value for the abstraction is higher than the mean value

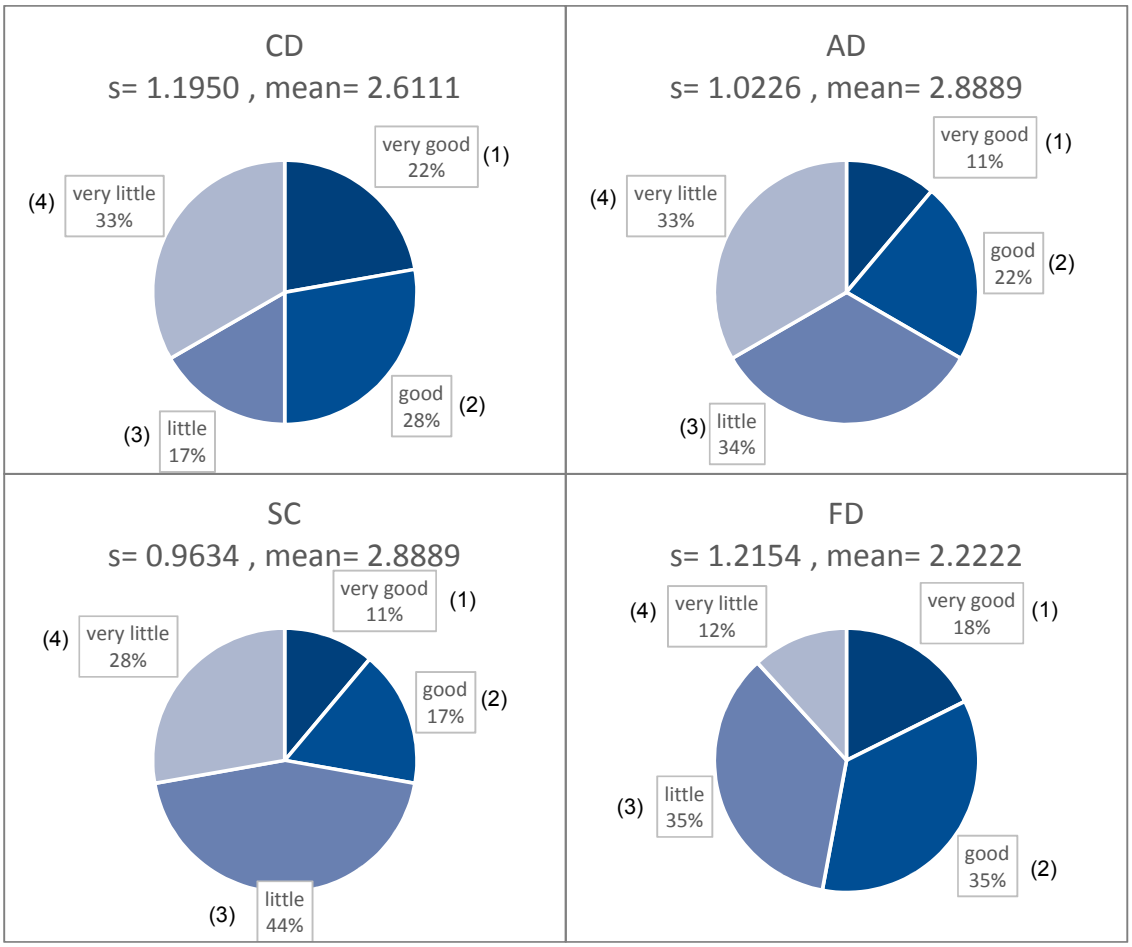


Figure 4.7: Percentages of the answers to Q B.5-E.5 about the helpfulness of summarization for understanding model differences.

for the semantic difference. For ADs, CDs, and FDs, the numbers of participants who considered the semantic difference very helpful was higher than the number of participants who consider the abstraction very helpful. The results also indicate that abstraction, in case of CDs, ADs, and FDs, is less helpful to provide explanations of model differences than the combination. For SCs, however, the results indicate that the abstraction is slightly more helpful to provide a helpful explanation of differences. Here, 61% of all participants considered the abstraction at least helpful, whereas for the combination, it was only 50%. Still, the mean values do not differ significantly.

4.6 Summarization of Semantic Differences

Figure 4.7 shows the answers to the question about how helpful the participants consider a summarization of the semantic differences to understand model differences.

For CDs, 22% considered the summarization very helpful, 28% considered it helpful, 17% considered it little helpful, and 33% found it very little helpful. The diagram at the top left of Figure 4.7 shows these results. For ADs, 11% found the summarization very helpful, 22% found it helpful, 34% considered it little helpful, and 33% of the participants considered

the summarization to help very little for understanding AD differences (cf. diagram at the top left of Figure 4.7). To understand SC differences, 11% considered the summarization very helpful, 17% considered it helpful, 44% little helpful, and 28% very little helpful, which the diagram at the bottom left of Figure 4.7 shows. For FDs, the summarization was considered very helpful by 18% of the participants, helpful by 35%, little helpful by 35%, and very little helpful by 12%.

Conclusion The results indicate that hypothesis H5 is incorrect regarding the syntactic difference, as it was considered at least helpful by more than 90% of all participants regardless of the modeling language.

Regarding the semantic difference, for CDs, ADs, and FDs, the semantic difference by itself was considered more helpful than the summarization. The mean values are significantly higher regarding the helpfulness of the summarization. For CDs and ADs, the mean is close to three, meaning that, in average, participants found the summarization little helpful. For FDs, the mean is close to two, therefore the summarization was, on average, considered helpful by the participants, however, the number of participants who considered the semantic FD difference very helpful (35%) is significantly lower for the summarization (18%). For SCs, the results indicate that the summarization is roughly equally helpful than the semantic difference. Comparing the results of the combination to the results of the summarization, we conclude that the combination is better suited to provide explanations for model differences regardless of the modeling language. For CDs the summarization was considered less helpful than the abstraction since the number of participants who found the summarization to be very little helpful has increased by 27% compared to the abstraction. For ADs, the summarization and abstraction are roughly equally helpful to provide explanations for model differences. For FDs, the number of participants who considered the summarization at least helpful (53%) is much higher than that of those who consider the abstraction at least helpful (30%). Therefore, the summarization is better suited to provide explanations for FD differences. For SCs, the summarization is considered much less helpful to explain differences than the abstraction, since 72% consider the summarization little to very little helpful.

4.7 Use Cases for Model Differencing

The last question openly asked which use cases the participants could imagine for applying the presented techniques for semantic and syntactic differences. Figure 4.8 visualizes the 39 answers as a word cloud. We categorized the results into four classes, i.e., analysis, bug handling, change management, explanation, and testing. Figure 4.9 shows the absolute numbers of answers belonging to each category. Since the participants were not restricted at all in giving their answers, many participants gave more than one suggestion. Therefore, the sample size is higher for this question. Clearly, most of the answers considered an application of the techniques for syntactic and semantic differencing in change management, closely followed by answers regarding explanations.

Change management Figure 4.10 shows a word cloud of the answers in the change management category. In this category, thirteen answers were given in total. The answers

Figure 4.8: Word cloud of the answers to Q F.1 about possible use cases to apply techniques for identifying syntactic and semantic differences.

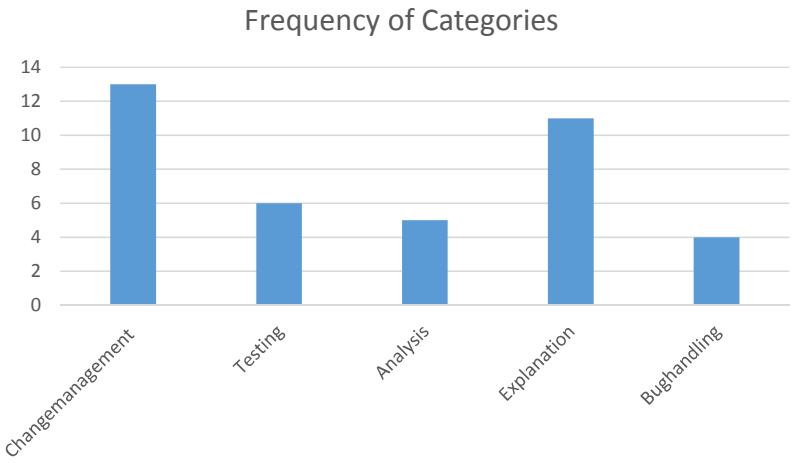


Figure 4.9: Numbers of answers for each category.

Figure 4.10: Word cloud of the answers in the change management category.

Figure 4.11: Word cloud of the answers in the explanation category.

Figure 4.12: Word cloud of the answers in the testing category.

often referenced semantic and syntactic differencing to be applicable in version control systems. Merging was often explicitly or implicitly regarded as extensible by the identification of syntactic as well as semantic differences.

Explanation Figure 4.11 shows a word cloud of the answers in the explanation category, comprising eleven answers. Many of our participants considered the techniques valuable to provide detailed illustrations of models, differences, and model-updates. Participants suggested to utilize the illustrations for documentation, illustration, or teaching purposes. Especially in distributed development environments, differencing methods were regarded helpful for understanding updates or differences between model versions and their impact on the modeled system.

Testing Figure 4.12 shows a word cloud of the given answers in the testing category, which comprised six answers in total. Surprisingly, testing was also regarded as a possible area of application for semantic and syntactic differencing. The generation of test cases or the adaptation of test cases by changing models was considered possible by utilizing

Figure 4.13: Word cloud of the answers in the analysis category.

Figure 4.14: Word cloud of the answers in the bug handling category.

syntactic and/or semantic model differences. Also the verification of requirements after model changes was mentioned in two answers of this category.

Analysis Figure 4.13 shows a word cloud of the answers in the analysis category which comprised five answers in total. Some of the participants considered the presented techniques for semantic and syntactic differencing for application in, e.g., requirements or cost analysis. Implicitly, the participants referred to automating the retrieval of certain information from the models by utilizing semantic and/or syntactic differencing, e.g., which processes or "constellations" were removed or added due to a change operation on a model.

Bug handling Figure 4.14 shows a word cloud of the answers in the bug handling category which comprised four answers in total. A few of our participants considered syntactic and/or semantic differencing useful for (automatic) bug fixing or error-correction.

It was considered particularly interesting to verify that an intended refactoring did not change the semantics of the modeled system.

4.8 Conclusion

The participants consider the syntactic difference most helpful to provide an explanation for model differences, regardless of the modeling language. The results contradict the hypothesis H2, stating that the semantic difference provides more intuitive explanations for model differences, regardless of the modeling language. A combination of syntactic and semantic differencing seems to be the most suitable alternative to providing intuitive explanations that take semantic differences into consideration. Regarding H6, the categories into which we grouped the answers of the participants suggest that there were five areas of application for syntactic and semantic differencing. Unsurprisingly, many participants suggested to apply model differencing in version control systems.

4.9 Result Discussion and Threads to Validity

Generally, the number of participants who conducted the survey was too low to provide statistically meaningful results. The low sample size also prevents a meaningful and significant correlation analysis. Regarding the relatively low levels of modeling experience, the results apply mainly for modelers with less than five years of experience and are not significant for this group due to the low sample size. Since most of the participants were working in the research domain, the results are also not applicable to modeling in the industrial practice. The lengthy explanations could have fostered random answers by the participants, which also threatens the validity of the study. Clearly, to work with models on paper instead of screens has an influence on the quality of the evaluation, especially if software engineers are used to work with models on screen, which again threatens the validity of the study.

We cannot generally conclude from the results that the syntactic difference is always the better choice, since we did not ask whether the syntactic difference would have provided sufficient understanding of the actual differences to the user. Possibly, the participants consider the semantic difference to accompany the syntactic difference in a way that provides a deeper understanding of the model differences. To test this, future studies should include a question asking whether the participants find the semantic difference and the syntactic difference to complement each other.

Furthermore, the example models are relatively small compared to real-world models. With increasing model size, the syntactic difference can become much more confusing. Moreover, the results reflect only one moment in time. In order to obtain more general results, a long-term study with users, their everyday used modeling language, and larger models would be necessary. To integrate the model differencing operators in version control systems and to conduct the study accompanied in an online version could also help to get more meaningful results.

Chapter 5

Related Work

This section presents related work on approaches for semantic differencing and on approaches for combining syntactic with semantic differencing. The related approaches can be potentially used for future evaluations. We are not aware of other studies investigating the usefulness of semantic differencing for developers.

Existing methods for semantic differencing can be categorized into enumerative and non-enumerative approaches [LMK14]. An enumerative approach (e.g., [MRR11f, MRR11b, BKRW17, KR18b, AHC+12]) takes two models as input. It outputs a finite set of diff witnesses. This survey solely includes enumerative approaches. Non-enumerative approaches (e.g., [FLW11, FALW14]) also take two models as input. Instead of returning witnesses, an non-enumerative approach returns a model that summarizes semantic differences (not necessarily all) from the one model to the other model.

This survey included enumerative semantic differencing approaches for CDs [MRR11d, KMRR17], ADs [KR18b], SCs [DEKR19], and FDs [DKMR19].

Another enumerative semantic differencing operator for ADs is described in [MRR11b]. As in the approach included in this survey for AD differencing [KR18b], the set of models in the approach presented in [MRR11b] is the set of all valid ADs, the semantic domain is the set of all possible finite execution traces over actions labels, and the semantic mapping maps each AD to the set of execution traces that it describes. The semantic differencing operator of [MRR11b] is grounded in an algorithm executing a fixed point calculation inspired by symbolic model-checking [BCM+92].

An enumerative semantic differencing operator for a feature model language is presented in [AHC+12]. The semantic mapping is based on the closed-world feature model semantics that is also used for the survey presented in this report. The semantics of a feature model solely contains configurations containing features that are used in the feature model. The semantic differencing operator uses a translation to the boolean satisfiability problem for propositional logic.

Another approach to semantic feature model differencing is based on an open-world semantics for feature models [DKMR19]. The set of models and the semantic domain are the same as in the closed-world approaches. However, the configurations in the semantics of a feature model are allowed to contain features that are not used in the feature model. The features not used in a feature model are considered to be unconstrained regarding their selection for valid configurations. This semantics is motivated by analyses in early

development stages where potentially not all features are known a priori before developing a feature model. An implementation of the open-world semantic differencing operator can be achieved via a translation to the boolean satisfiability problem [DKMR19].

A semantic differencing operator for an automaton variant for modeling interactive reactive systems is presented in [BKRW17]. The models are called finite time-synchronous port automata (TSPAs). The semantic domain contains all possible behaviors. Each behavior models an infinite history of messages communicated via channels. The semantics of a TSPA is the set of all behaviors modeled by the TSPA. The semantic differencing operator translates the two input TSPAs to Büchi automata [Büc62, Far02, Saf88], before it checks whether the language accepted by the one Büchi automaton resulting from the translation is a subset of the language accepted by the other Büchi automaton resulting from the translation. Time-synchronous channel automata [BKRW19] are similar to TSPAs. A semantic differencing operator for this automaton variant and a subclass for which semantic differencing can be performed efficiently is described in [BKRW19].

A generic method to semantic differencing based on behavioral semantics specifications is described in [LMK14]. The method is instantiated with an AD, a CD, and a Petri net language. The method relies on executing models, capture execution traces, and compare the captured traces.

A method for semantic differencing of combinatorial models of test designs is described in [TBM17]. The models are combinatorial models. The semantic domain consists of tests. Each test consists of parameter value assignments. Each combinatorial model consists of constraints regarding the values that are assignable to parameters. Each element in the semantic domain is called a test. The semantics of a combinatorial model contains all tests assigning values to parameters such that the constraints of the model are satisfied. The approach presents semantic differences in the form of strongest exclusions. Strongest exclusion are (smallest) parameter value assignment combinations excluded by the constraints of a combinatorial model. The semantic difference could also be presented by diff witnesses representing complete tests [TBM17]. However, as stated in [TBM17], this representation may have semantic problems [TBM17].

Previous works applied non-enumerative semantic differencing approaches to feature models [FLW11], automata [FLW11], and CDs [FALW14]. With non-enumerative semantic differencing approaches, semantic differences are represented by models and not by diff witnesses.

The approaches presented in [FLW11, FALW14] rely on the existence of composition operators satisfying special properties. A differencing result is a model such that composing one of the input models with the differencing results yields a model that does not exhibit any semantic differences to the other model.

The Diffuse framework [MR15, MR18] defines analyses that take two models, a diff witness, and syntactic changes that change the one model to the other model as input. One analysis can be used to compute subsets of the syntactic changes such that the application of the subset guarantees the existence of the diff witness. Another analysis can be used to compute subsets of the syntactic changes such that the application of the subset to the one model would not cause the existence of the witness. The analyses enabled by Diffuse [MR15, MR18] consider a concrete changelog, whereas the framework for computing repairing change sequences [KR18a] determines changes without considering a changelog.

Chapter 6

Conclusion

This technical report presented the results of a survey among software engineers and students. The survey has been conducted for evaluating the ability of semantic differencing techniques to explain and visualize differences between model versions to developers. The survey investigated the usefulness of model differencing operator results with respect to the comprehensibility of the differences between model versions. We have investigated the comprehensibility of the results computed by differencing operators for Class Diagrams, Activity Diagrams, Statecharts and Feature Diagrams.

Chapter 2 described a generic framework for model differencing and the instantiations with the concrete modeling languages. Chapter 3 explained the design of the study in detail. Chapter 4 presented the results of the study and our main findings. Chapter 5 discussed related work.

The study participants were mostly unexperienced, with zero to at most five years of modeling experience. For all considered languages, more than 90% of the participants considered the syntactic difference very helpful or helpful to understand the model differences. The participants mostly considered syntactic differencing results to be more helpful than semantic differencing results. Whether techniques combining semantic and syntactic differencing are more helpful than semantic differencing operators in isolation seems to be language-dependent. The results from semantic differencing operators without applying abstraction techniques seem to be mostly at least as helpful as the results computed by semantic differencing operators while using abstractions. The results computed by semantic differencing operators without the application of summarization techniques were considered to be more helpful than summarized results. However, the number of participants who conducted the survey was too low to provide meaningful results and the low sample size also prevents a meaningful and significant correlation analysis.

It cannot be generally concluded from the results of the survey that presenting syntactic differences is always more beneficial than presenting semantic differences. The survey did not include questions asking whether the syntactic difference would have provided a sufficient understanding of the actual differences. It could be possible that the participants considered the semantic difference to accompany the syntactic difference in a way that provides a deeper understanding of the model differences. Furthermore, the example models used as basis for the questions are all relatively small. For large models, the syntactic difference is often more confusing than for small models. Then, understanding the semantic model differences by solely examining the syntactic differences is usually

complicated. In contrast, for small models, the syntactic difference is often small and easy to understand. Then, it is relatively easy to understand the semantic model differences by solely examining the syntactic differences. We conclude that further surveys with more and larger models and more participants are necessary to obtain generalizable results.

Bibliography

[AHC+12] Mathieu Acher, Patrick Heymans, Philippe Collet, Clément Quinton, Philippe Lahire, and Philippe Merle. Feature Model Differences. In Jolita Ralyté, Xavier Franch, Sjaak Brinkkemper, and Stanislaw Wrycza, editors, *Advanced Information Systems Engineering*, pages 629–645. Springer Berlin Heidelberg, 2012.

[AP03] Marcus Alanen and Ivan Porres. Difference and Union of Models. In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *UML 2003 - The Unified Modeling Language. Modeling Languages and Applications*, pages 2–17. Springer Berlin Heidelberg", 2003.

[ASW09] Kerstin Altmanninger, Martina Seidl, and Manuel Wimmer. A survey on model versioning approaches. *International Journal of Web Information Systems*, 5(3):271–304, 2009.

[BCM+92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation*, 98(2):142–170, 1992.

[BCW17] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice: Second Edition.* Morgan & Claypool Publishers, 2nd edition, 2017.

[BKL+12] Petra Brosch, Gerti Kappel, Philip Langer, Martina Seidl, Konrad Wieland, and Manuel Wimmer. An Introduction to Model Versioning. In Marco Bernardo, Vittorio Cortellessa, and Alfonso Pierantonio, editors, *Formal Methods for Model-Driven Engineering: 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, pages 336–398. Springer Berlin Heidelberg, 2012.

[BKRW17] Arvid Butting, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. Semantic Differencing for Message-Driven Component & Connector Architectures. In *International Conference on Software Architecture (ICSA'17)*, pages 145–154. IEEE, 2017.

[BKRW19] Arvid Butting, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. Continuously analyzing finite, message-driven, time-synchronous component & connector systems during architecture evolution. *Journal of Systems and Software*, 149:437–461, 2019.

[Büc62] Julius Richard Büchi. On a decision method in restricted second order arithmetic. In *International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11, 1962.

[CGR09] María Victoria Cengarle, Hans Grönniger, and Bernhard Rumpe. Variability within Modeling Language Definitions. In Andy Schürr and Bran Selic, editors, *Model Driven Engineering Languages and Systems*, pages 670–684. Springer Berlin Heidelberg, 2009.

[DEKR19] Imke Drave, Robert Eikermann, Oliver Kautz, and Bernhard Rumpe. Semantic Differencing of Statecharts for Object-Oriented Systems. In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development*, pages 274–282. SciTePress, 2019.

[DKMR19] Imke Drave, Oliver Kautz, Judith Michael, and Bernhard Rumpe. Semantic Evolution Analysis of Feature Models. In *Proceedings of the 23rd International Systems and Software Product Line Conference*, page 245–255. ACM, 2019.

[EPK06] Klaus-D. Engel, Richard F. Paige, and Dimitrios S. Kolovos. Using a Model Merging Language for Reconciling Model Versions. In Arend Rensink and Jos Warmer, editors, *Model Driven Architecture – Foundations and Applications*, pages 143–157. Springer Berlin Heidelberg, 2006.

[FALW14] Uli Fahrenberg, Mathieu Acher, Axel Legay, and Andrzej Wasowski. Sound Merging and Differencing for Class Diagrams. In Stefania Gnesi and Arend Rensink, editors, *Fundamental Approaches to Software Engineering*, pages 63–78. Springer Berlin Heidelberg, 2014.

[Far02] Berndt Farwer. ω-automata. In Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors, *Automata Logics, and Infinite Games: A Guide to Current Research*, pages 3–21. Springer Berlin Heidelberg, 2002.

[FLW11] Uli Fahrenberg, Axel Legay, and Andrzej Wasowski. Vision Paper: Make a Difference! (Semantically). In Jon Whittle, Tony Clark, and Thomas Kühne, editors, *Model Driven Engineering Languages and Systems*, pages 490–500. Springer Berlin Heidelberg, 2011.

[FR07] Robert France and Bernhard Rumpe. Model-Driven Development of Complex Software: A Research Roadmap. In *Future of Software Engineering (FOSE '07)*, pages 37–54, 2007.

[GKLE13] Christian Gerth, Jochen M. Küster, Markus Luckey, and Gregor Engels. Detection and resolution of conflicting change operations in version management of process models. *Software & Systems Modeling*, 12(3):517–535, 2013.

[GR11] Hans Grönniger and Bernhard Rumpe. Modeling Language Variability. In Radu Calinescu and Ethan Jackson, editors, *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, pages 17–32. Springer Berlin Heidelberg, 2011.

[Grö10] Hans Grönniger. *Systemmodell-basierte Definition objektbasierter Modellierungssprachen mit semantischen Variationspunkten.* Aachener Informatik-Berichte, Software Engineering, Band 4. Shaker, August 2010.

[HKR+07] Christoph Herrmann, Holger Krahn, Bernhard Rumpe, Martin Schindler, and Steven Völkel. An Algebraic View on the Semantics of Model Composition. In David H. Akehurst, Régis Vogel, and Richard F. Paige, editors, *Model Driven Architecture- Foundations and Applications*, pages 99–113. Springer Berlin Heidelberg, 2007.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[HR00] David Harel and Bernhard Rumpe. Modeling Languages: Syntax, Semantics and All That Stuff (Part I: The Basic Stuff). Technical Report MCS00-16, Mathematics & Computer Sience, Weizmann Institute Of Sience, 2000.

[HR04] David Harel and Bernhard Rumpe. Meaningful Modeling: What's the Semantics of "Semantics"? *Computer*, 37(10):64–72, 2004.

[Jac06] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006.

[KKT13] Timo Kehrer, Udo Kelter, and Gabriele Taentzer. Consistency-preserving edit scripts in model versioning. In *IEEE/ACM International Conference on Automated Software Engineering (ASE'13)*, pages 191–201. IEEE, 2013.

[KMRR17] Oliver Kautz, Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. CD2Alloy: A Translation of Class Diagrams to Alloy. Technical Report AIB-2017-06, RWTH Aachen University, July 2017.

[KR18a] Oliver Kautz and Bernhard Rumpe. On Computing Instructions to Repair Failed Model Refinements. In *MODELS'18: Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 289–299. ACM, 2018.

[KR18b] Oliver Kautz and Bernhard Rumpe. Semantic Differencing of Activity Diagrams by a Translation into Finite Automata. In *Proceedings of MODELS 2018. Workshop ME*, October 2018.

[KRR18] Oliver Kautz, Alexander Roth, and Bernhard Rumpe. *Achievements, Failures, and the Future of Model-Based Software Engineering*, pages 221–236. Springer International Publishing, June 2018.

[LMK14] Philip Langer, Tanja Mayerhofer, and Gerti Kappel. Semantic Model Differencing Utilizing Behavioral Semantics Specifications. In Juergen Dingel, Wolfram Schulte, Isidro Ramos, Silvia Abrahão, and Emilio Insfran, editors, *Model-Driven Engineering Languages and Systems*, pages 116–132. Springer International Publishing, 2014.

[MD08] Tom Mens and Serge Demeyer, editors. *Software Evolution*. Springer, 2008.

[MGH05] Akhil Mehra, John Grundy, and John Hosking. A Generic Approach to Supporting Diagram Differencing and Merging for Collaborative Design. In *International Conference on Automated Software Engineering*, 2005.

[MR15] Shahar Maoz and Jan Oliver Ringert. A Framework for Relating Syntactic and Semantic Model Differences. In *Proceedings of the 18th International Conference on Model Driven Engineering Languages and Systems*, page 24–33. IEEE, 2015.

[MR18] Shahar Maoz and Jan Oliver Ringert. A framework for relating syntactic and semantic model differences. *Software & Systems Modeling*, 17(3):753–777, 2018.

[MRR11a] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. A Manifesto for Semantic Model Differencing. In Juergen Dingel and Arnor Solberg, editors, *Models in Software Engineering*, pages 194–203. Springer Berlin Heidelberg, 2011.

[MRR11b] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. ADDiff: Semantic Differencing for Activity Diagrams. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, pages 179–189. ACM, 2011.

[MRR11c] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. CD2Alloy: Class Diagrams Analysis Using Alloy Revisited. In *Proceedings of the 14th International Conference on Model Driven Engineering Languages and Systems*, pages 592–607. Springer, 2011.

[MRR11d] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. CDDiff: Semantic Differencing for Class Diagrams. In Mira Mezini, editor, *ECOOP 2011 – Object-Oriented Programming*, pages 230–254. Springer Berlin Heidelberg, 2011.

[MRR11e] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Semantically Configurable Consistency Analysis for Class and Object Diagrams. In Jon Whittle, Tony Clark, and Thomas Kühne, editors, *Model Driven Engineering Languages and Systems*, pages 153–167. Springer Berlin Heidelberg, 2011.

[MRR11f] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Summarizing Semantic Model Differences. In *Models and Evolution Workshop (ME'17) at MODELS*, 2011.

[MRR12] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. An Interim Summary on Semantic Model Differencing. *Softwaretechnik-Trends*, 32(4):44–46, 2012.

[OMW05] Hamilton L. R. Oliveira, Leonardo Gresta Paulino Murta, and Cláudia Werner. Odyssey-VCS: A Flexible Version Control System for UML Model Elements. In *Proceedings of the 12th International Workshop on Software Configuration Management*, pages 1–16. ACM, 2005.

[OWK03] Dirk Ohst, Michael Welle, and Udo Kelter. Differences between versions of UML diagrams. In *Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, page 227–236. ACM, 2003.

[Rum98] Bernhard Rumpe. A Note on Semantics (with an Emphasis on UML). In *Proceedings Second ECOOP Workshop on Precise Behavioral Semantics (with an*

Emphasis on OO Business Specifications). Technische Universität München, TUM-I9813, 1998.

[Saf88] Shmuel Safra. On The Complexity of ω-Automata. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 319–327. IEEE, 1988.

[Sch06] Douglas C. Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):25–31, 2006.

[Sel03] Bran Selic. The pragmatics of model-driven development. *IEEE Software*, 20(5):19–25, 2003.

[TBM17] Rachel Tzoref-Brill and Shahar Maoz. Syntactic and Semantic Differencing for Combinatorial Models of Test Designs. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 621–631. IEEE, 2017.

[TELW14] Gabriele Taentzer, Claudia Ermel, Philip Langer, and Manuel Wimmer. A fundamental approach to model versioning based on graph modifications: from theory to implementation. *Software & Systems Modeling*, 13(1):239–272, 2014.

[XS07] Zhenchang Xing and Eleni Stroulia. Differencing logical UML models. *Automated Software Engineering*, 14(2):215–259, 2007.

Appendix A

Study Questionnaire

Studie zum syntaktischen und semantischen Unterschied von Modellen

Liebe Studienteilnehmerinnen und –teilnehmer,

die Verwendung von modellbasierten Methoden im Entwicklungsprozess stellt die Entwicklerinnen und Entwickler vor die Herausforderung, dass Sie mit Änderungen in Modellen umgehen und diese verstehen müssen. Hierfür gibt es unterschiedliche Methoden, die die Unterschiede zwischen Modellversionen berechnen.

Diese Studie beschäftigt sich mit Möglichkeiten zur Identifikation von syntaktischen und semantischen Unterschieden zwischen Modellen in vier Modellierungssprachen: Klassendiagramme, Aktivitätsdiagramme, State Charts und Feature-Diagramme.

Die Beantwortung dieser Fragen ermöglicht es uns Rückschlüsse auf das Verständnis der Ansätze ziehen zu können. Die Ergebnisse werden im Rahmen des DFG Projekts „A Semantic Approach to Evolution Analysis in Model-Based Software Development“ wissenschaftlich verwendet.

Herzlichen Dank für Ihre Teilnahme!

Imke Drave, Oliver Kautz und Judith Michael

Software Engineering, RWTH Aachen
{nachname}@se-rwth.de

Block A: Allgemeine Fragen

Frage A.1: Wie hoch schätzen sie ihre Expertise mit den folgenden Modellierungssprachen ein?

Klassendiagramme

☐	☐	☐	☐
Sehr hoch	eher hoch	eher niedrig	sehr niedrig

Aktivitätsdiagramme

☐	☐	☐	☐
Sehr hoch	eher hoch	eher niedrig	sehr niedrig

State Charts

☐	☐	☐	☐
Sehr hoch	eher hoch	eher niedrig	sehr niedrig

Feature-Diagramme

☐	☐	☐	☐
Sehr hoch	eher hoch	eher niedrig	sehr niedrig

Frage A.2: Wie lange modellieren sie schon?

☐	☐	☐	☐
0 Jahre	>0 bis <= 5 Jahre	>5 bis <= 10 Jahre	> 10 Jahre

Unterschiede zwischen Klassendiagrammen

Im Folgenden wird ein Scenario zur Evolutionsanalyse von Klassendiagrammen vorgestellt. Im Scenario werden zwei Klassendiagramme gezeigt. Das mit v2 gekennzeichnete Klassendiagramm ist die Nachfolgerversion des mit v1 gekennzeichneten Klassendiagramms. Es werden der syntaktische Unterschied und der semantische Unterschied des Klassendiagramms mit der Version v2 zum Klassendiagramm mit der Version v1 erläutert.

Versuchen Sie bitte die erläuterten Unterschiede nachzuvollziehen. Bewerten Sie bitte jeweils anschließend auf dieser Grundlage inwiefern Ihnen die Erläuterungen beim Verständnis der Unterschiede von der Nachfolgerversion v2 zur Version v1 geholfen haben.

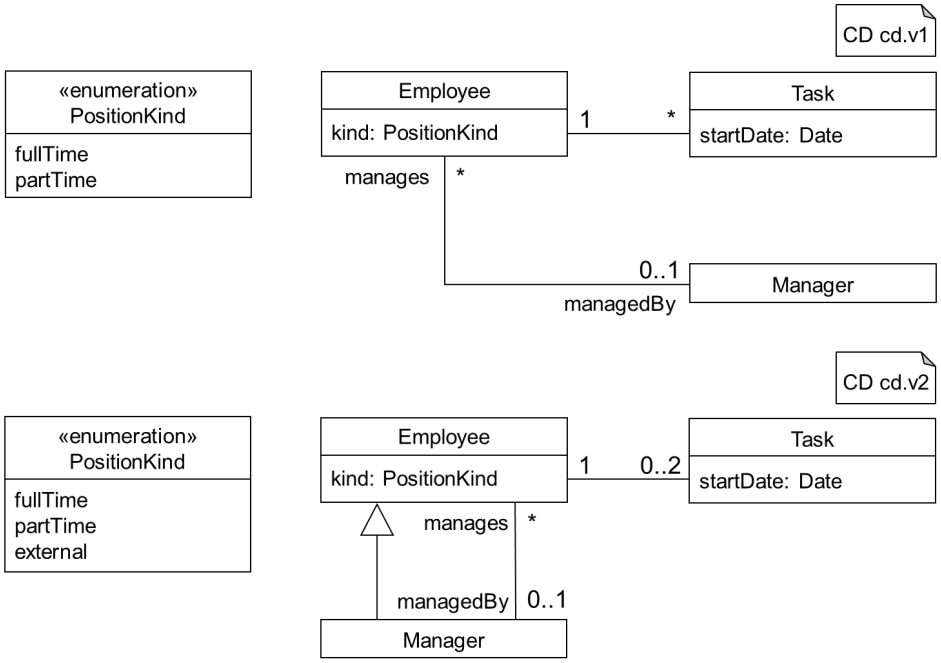
Block B: UML/P Klassendiagramme

Klassendiagramm Scenario

Sie modellieren die Personal-Struktur eines Unternehmens, das Voll- und Teilzeitangestellte hat. Jeder Angestellte ist einem oder keinem Manager unterstellt. Jedem Angestellten können Aufgaben zugewiesen werden. Eine Aufgabe muss mit einem Startdatum gekennzeichnet werden. Manager können beliebig viele Angestellte managen. Sie erstellen das Klassendiagramms cd.v1.

Die Struktur im Unternehmen wurde im Zuge von Maßnahmen zur Verbesserung der Work-Life-Balance und der allgemeinen Zufriedenheit verändert. Ein Arbeitskollege ändert daraufhin das Klassendiagramm cd.v1.

Daraus entsteht das Klassendiagramm cd.v2. Sie möchten die Unterschiede vom neuen Klassendiagramm cd.v2 zum Klassendiagramm cd.v1 verstehen.



Block B: UML/P Klassendiagramme

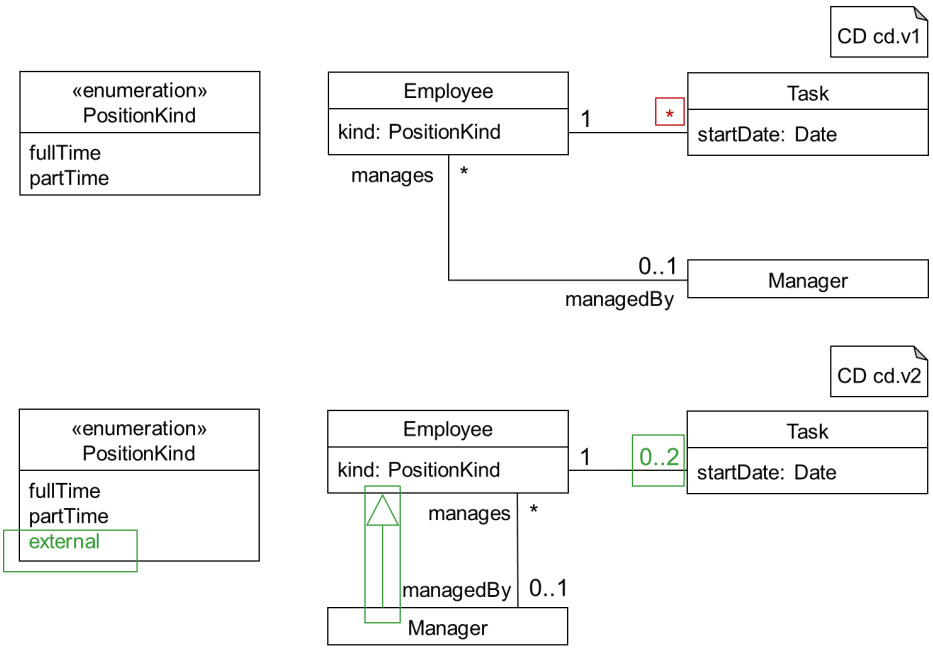
Syntaktische Unterschiede im Klassendiagramm Scenario (B1)

Erläuterung des syntaktischen Unterschieds von cd.v2 zu cd.v1 in textueller Form:

1. Am Assoziationsende bei der Klasse Task der Assoziation zwischen den Klassen Employee und Task wurde die Kardinalität von * auf 0..2 geändert.
2. Es wurde eine Spezialisierungsbeziehung von der Klasse Manager zur Klasse Employee hinzugefügt.
3. Es wurde der Wert external zur Enumeration PositionKind hinzugefügt.

Erläuterung des syntaktischen Unterschieds in grafischer Form:

Die im folgenden Bild im Klassendiagramm cd.v1 rot dargestellten und durch einen Kasten markierten Elemente wurden gelöscht oder geändert. Die im Klassendiagramm cd.v2 grün dargestellten und durch einen Kasten markierten Elemente wurden hinzugefügt oder geändert.

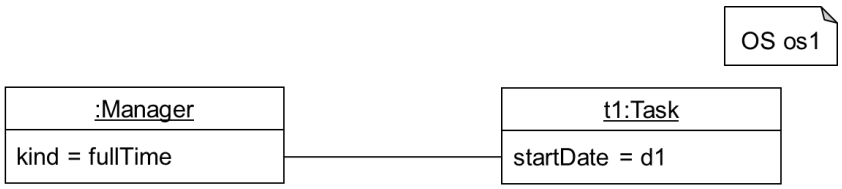


Semantische Unterschiede im Klassendiagramm Scenario

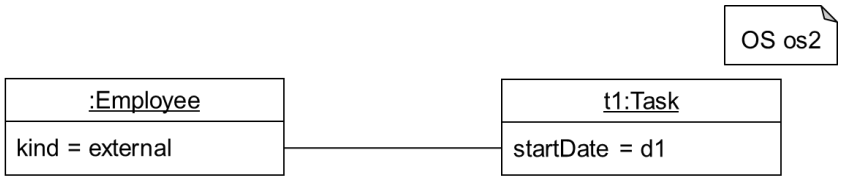
Erläuterung des semantischen Unterschieds (B2):

Die folgenden Objektstrukturen os1, os2, os3, os4, os5 und os6 sind Instanzen des Klassendiagramms cd.v2 und keine Instanzen des Klassendiagramms cd.v1:

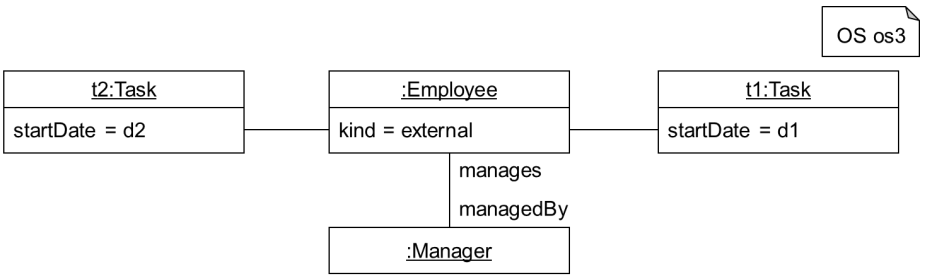
1. Objectstruktur:



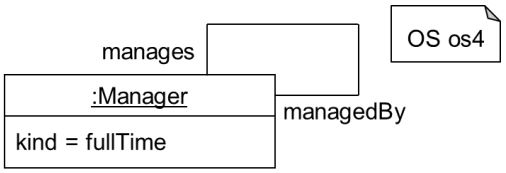
2. Objektstruktur:



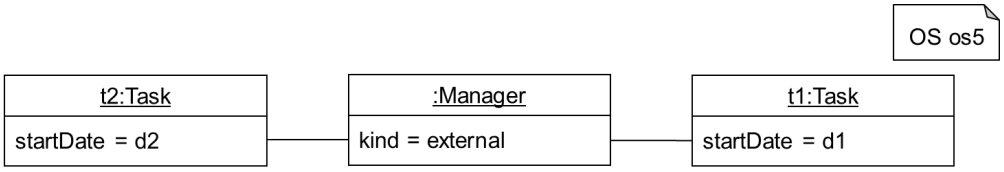
3. Objektstruktur:



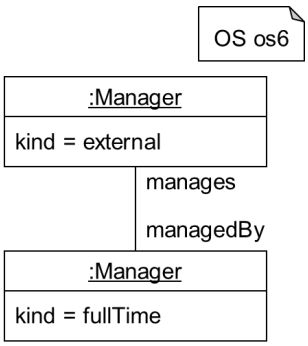
4. Objektstruktur:



5. Objektstruktur:



6. Objektstruktur:



Erläuterung des Unterschieds durch eine Kombination der syntaktischen Unterschiede mit den semantischen Unterschieden (B3):

Wenn die folgenden Änderungen am Klassendiagramm cd.v2 durchgeführt werden, dann erhält man ein Klassendiagramm, das ausschließlich Instanzen hat, die auch Instanzen des Klassendiagramm cd.v1 sind:

1. Die Spezialisierungsbeziehung von der Klasse Manager zur Klasse Employee entfernen.
2. Den Wert external von der Enumeration PositionKind entfernen.

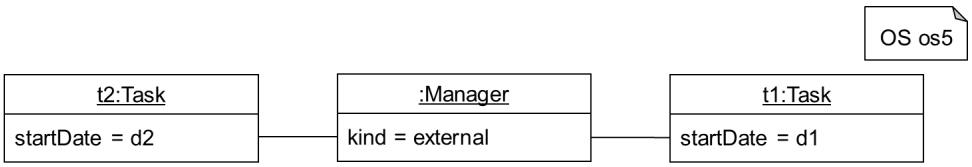
Erläuterungen des semantischen Unterschieds durch Beispielabstraktion (B4):

1. Wenn von der Spezialisierungsbeziehung von der Klasse Manager zur Klasse Employee und von dem Wert external der Enumeration PositionKind abstrahiert würde, dann wäre jede Instanz des Klassendiagramms cd.v2 auch eine Instanz des Klassendiagramms cd.v1.

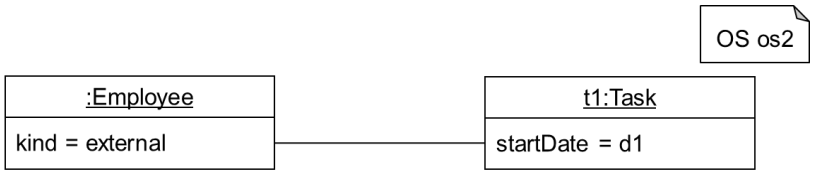
Erläuterung des semantischen Unterschieds durch Zusammenfassung (B5):

Die folgende Liste von Objektstrukturen repräsentiert eine Menge von Instanzen von cd.v2, die keine Instanzen von cd.v1 sind. Die Mengen der Klassen der Objekte der Instanzen unterscheiden sich paarweise.

1. Objektstruktur:



2. Objektstruktur:



Block B: UML/P Klassendiagramme

Frage B.1: Wie gut hat Ihnen die Angabe des syntaktischen Unterschieds beim Verstehen der Unterschiede der Modelle geholfen?

☐ Sehr gut ☐ eher gut ☐ eher wenig ☐ sehr wenig

Frage B.2: Wie gut hat Ihnen die Angabe des semantischen Unterschieds beim Verstehen der Unterschiede der Modelle geholfen?

☐ Sehr gut ☐ eher gut ☐ eher wenig ☐ sehr wenig

Frage B.3: Wie gut hat Ihnen die Erläuterung der Unterschiede durch eine Kombination der syntaktischen mit den semantischen Unterschieden beim Verstehen der Unterschiede der Modelle geholfen?

☐ Sehr gut ☐ eher gut ☐ eher wenig ☐ sehr wenig

Frage B.4: Wie gut hat Ihnen die Beispielabstraktion beim Verstehen der Unterschiede der Modelle geholfen?

☐ Sehr gut ☐ eher gut ☐ eher wenig ☐ sehr wenig

Frage B.5: Wie gut hat Ihnen die Zusammenfassung der semantischen Unterschiede beim Verstehen der Unterschiede der Modelle geholfen?

☐ Sehr gut ☐ eher gut ☐ eher wenig ☐ sehr wenig

Unterschiede zwischen Aktivitätsdiagrammen

Im Folgenden wird ein Scenario zur Evolutionsanalyse von Aktivitätsdiagrammen vorgestellt. Im Scenario werden zwei Aktivitätsdiagramme gezeigt. Das mit v2 gekennzeichnete Aktivitätsdiagramm ist die Nachfolgerversion des mit v1 gekennzeichneten Aktivitätsdiagramms. Es werden der syntaktische Unterschied und der semantische Unterschied des Aktivitätsdiagramms mit der Version v2 zum Aktivitätsdiagramm mit der Version v1 erläutert.

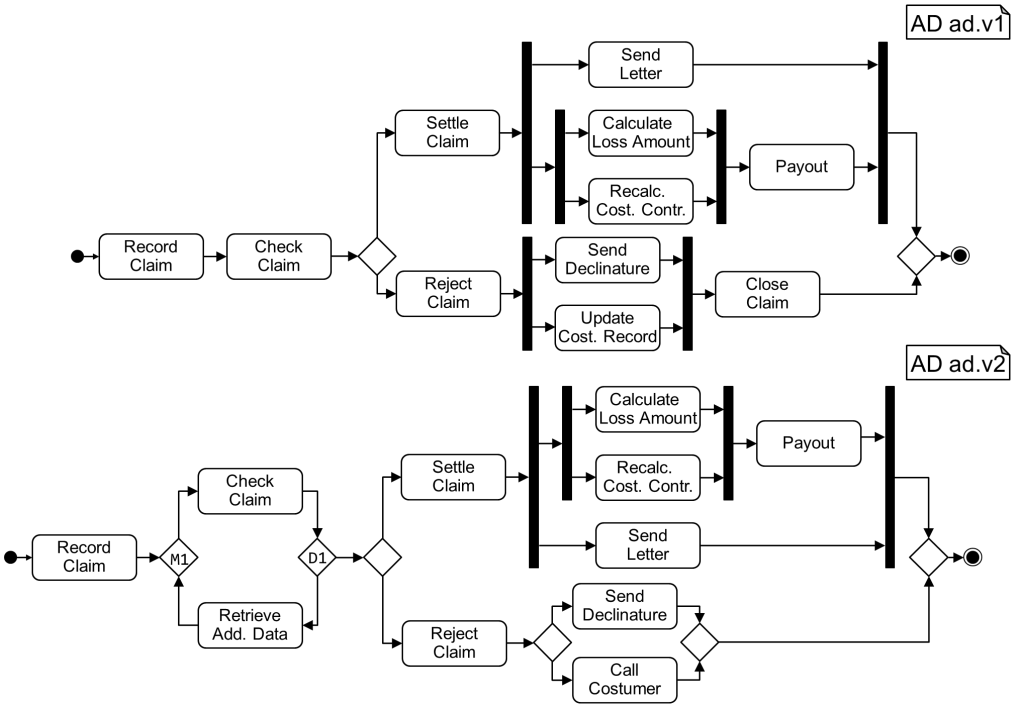
Versuchen Sie bitte die erläuterten Unterschiede nachzuvollziehen. Bewerten Sie bitte jeweils anschließend auf dieser Grundlage inwiefern Ihnen die Erläuterungen beim Verständnis der Unterschiede von der Nachfolgerversion v2 zur Version v1 geholfen haben.

Block C: UML/P Aktivitätsdiagramme

Aktivitätsdiagramm Scenario

Stellen Sie sich vor, Sie seien Mitarbeiter eines Versicherungsunternehmens. Die folgenden Aktivitätsdiagramme beschreiben das Vorgehen des Versicherungsunternehmens bei eingehenden Schadensmeldungen. Initial wurde das Vorgehen mit dem Aktivitätsdiagramm ad.v1 modelliert.

Nach einer Weile wird einem Ihrer Arbeitskollegen deutlich, dass das erste Aktivitätsdiagramm einige Sachverhalte nicht richtig abbildet. Ihr Arbeitskollege ändert daraufhin das Aktivitätsdiagramm. Daraus entsteht das Aktivitätsdiagramm ad.v2. Sie möchten die Unterschiede vom neuen Aktivitätsdiagramm ad.v2 zum Aktivitätsdiagramm ad.v1 verstehen.



Block C: UML/P Aktivitätsdiagramme

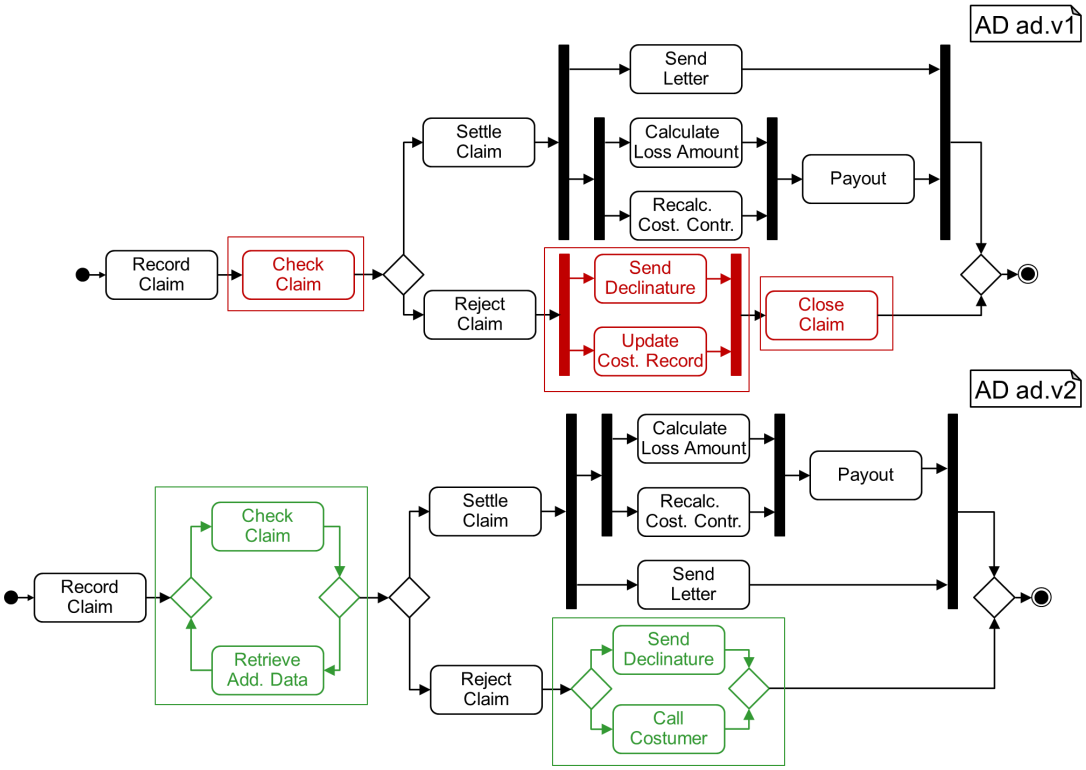
Syntaktische Unterschiede im Aktivitätsdiagramm Scenario (C1)

Erläuterung des syntaktischen Unterschieds von ad.v2 zu ad.v1 in textueller Form:

1. Die Aktion mit der Beschriftung CheckClaim wurde gelöscht.
2. Es wurde ein Zyklisches-Fragment zwischen dem Knoten mit der Beschriftung RecordClaim und dem darauffolgenden Decision Knoten hinzugefügt.
3. Die Aktion mit der Beschriftung CheckClaim wurde dem zyklischen-Fragment zwischen dem Merge und dem Decision Knoten hinzugefügt.
4. Die Aktion mit der Beschriftung Retrieve Add. Data wurde dem zyklischen-Fragment zwischen dem Decision und dem Merge Knoten hinzugefügt.
5. Das And-Fragment, das die Knoten mit den Beschriftungen Send Declinature und Update Cost. Record enthält, wurde zu einem Xor-Fragment umgewandelt.
6. Die Aktion mit der Beschriftung Update Cost. Record wurde gelöscht.
7. Die Aktion mit der Beschriftung Call Costumer wurde zum Xor-Fragment, das die Aktion mit der Beschriftung Send Declinature enthält, hinzugefügt.
8. Die Aktion mit der Beschriftung Close Claim wurde gelöscht.

Erläuterung des syntaktischen Unterschieds in grafischer Form:

Die im folgenden Bild im Aktivitätsdiagramm ad.v1 rot dargestellten und durch einen Kasten markierten Elemente wurden gelöscht oder geändert. Die im Aktivitätsdiagramm ad.v2 grün dargestellten und durch einen Kasten markierten Elemente wurden hinzugefügt oder geändert.



Semantische Unterschiede im Aktivitätsdiagramm Scenario

Erläuterung des semantischen Unterschieds in textueller Form (C2):

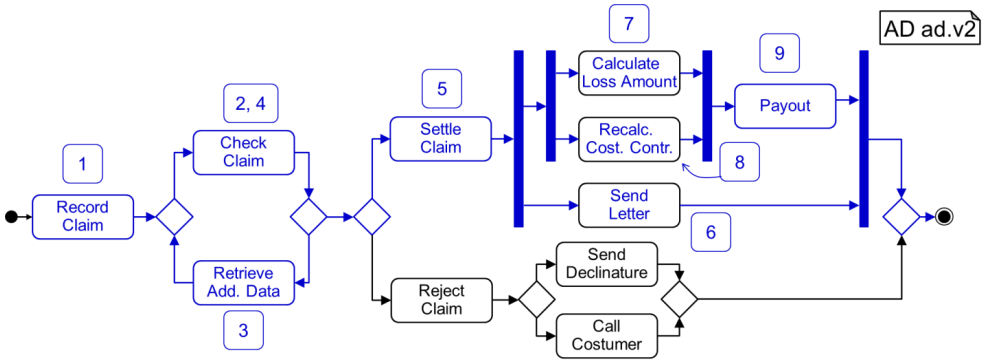
Die folgenden Abläufe sind im Aktivitätsdiagramm ad.v2 möglich und im Aktivitätsdiagramm ad.v1 nicht möglich:

1. Ablauf
   Record Claim, Check Claim, Retrieve Add. Data, Check Claim, Settle Claim, Send Letter, Calculate Loss Amount, Recalc. Cost. Contr., Payout.
2. Ablauf
   Record Claim, Check Claim, Retrieve Add. Data, Check Claim, Reject Claim, Send Declinature.
3. Ablauf:
   Record Claim, Check Claim, Retrieve Add. Data, Check Claim, Settle Claim, Calculate Loss Amount, Recalc. Cost. Contr., Send Letter, Payout.
4. Ablauf
   Record Claim, Check Claim, Reject Claim, Call Costumer.
5. Ablauf
   Record Claim, Check Claim, Retrieve Add. Data, Check Claim, Reject Claim, Call Costumer.

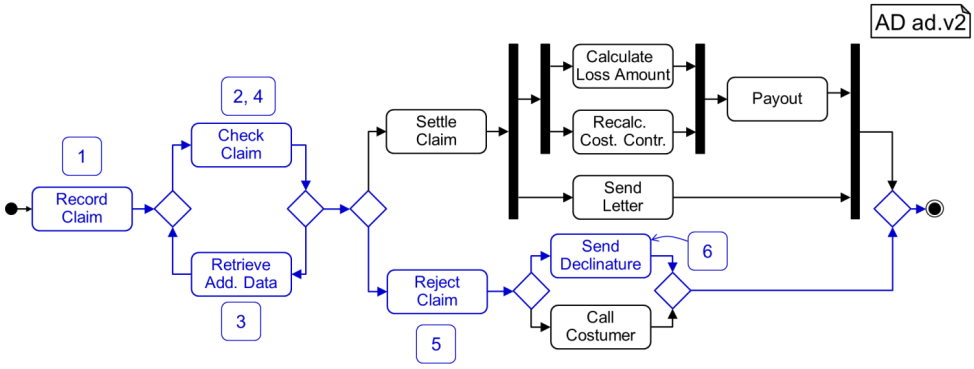
Erläuterung des semantischen Unterschieds in grafischer Form (C2):

Im den folgenden sechs Bildern werden die zuvor erläuterten Abläufe 1, 2, 3, 4, 5, 6 im Aktivitätsdiagramm ad.v2 grafisch dargestellt:

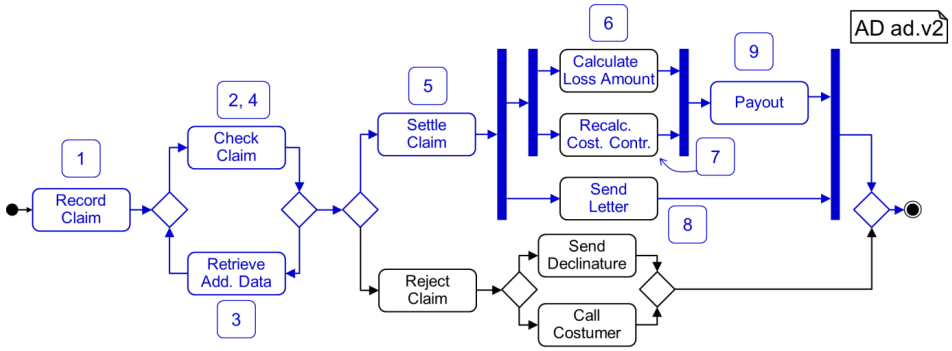
1. Ablauf:



2. Ablauf:

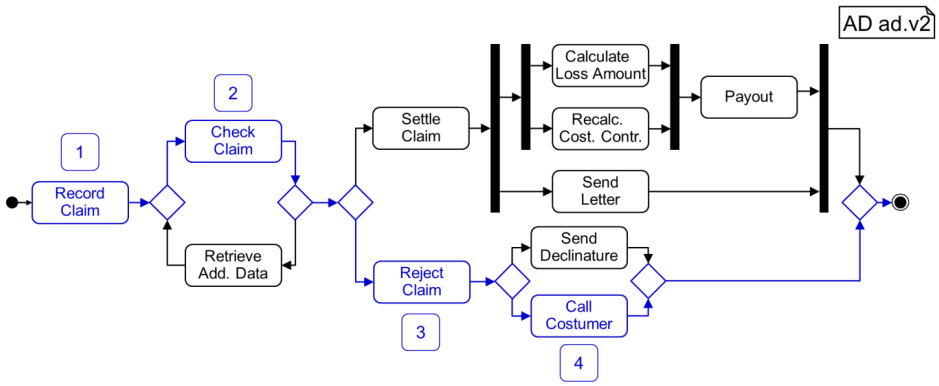


3. Ablauf:

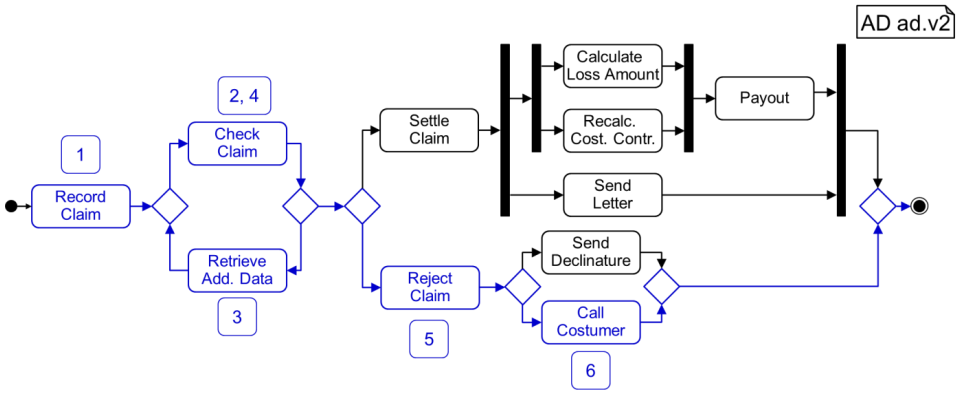


Block C: UML/P Aktivitätsdiagramme

4. Ablauf:



5. Ablauf:



Erläuterung des Unterschieds durch eine Kombination der syntaktischen Unterschiede mit den semantischen Unterschieden (C3):

Durch die Anwendung der folgenden Änderungen am Aktivitätsdiagramm ad.v2 erhält man ein Aktivitätsdiagramm, das ausschließlich Abläufe modelliert, die auch im Aktivitätsdiagramm ad.v1 modelliert sind:

1. Die Aktion mit der Beschriftung Retrieve Add. Data löschen.
2. Die Aktion mit der Beschriftung Call Costumer löschen.
3. Eine Aktion mit der Beschriftung Close Claim nach der Aktion mit der Beschriftung Send Declinature einfügen.
4. Eine Aktion mit der Beschriftung Update Cost. Record nach der Aktion mit der Beschriftung Send Declinature einfügen.

Erläuterungen des semantischen Unterschieds durch Beispielabstraktion (C4):

1. Wenn die Aktionen Retrieve Add. Data, Update Cost. Record, Call Costumer und Close Claim sowie die mit den Aktionen verbundenen Transitionen in den Aktivitätsdiagrammen nicht existieren würden, dann wäre jeder Ablauf von ad.v2 auch ein Ablauf von ad.v1.

Erläuterung des semantischen Unterschieds durch Zusammenfassung (C5):

Die folgende Liste repräsentiert eine Menge von Abläufen, die in ad.v2 möglich und in ad.v1 nicht möglich sind. Die Mengen der Aktionen aller Abläufe unterscheiden sich paarweise.

1. Ablauf
   Record Claim, Check Claim, Retrieve Add. Data, Check Claim, Reject Claim, Send Declinature.
2. Ablauf
   Record Claim, Check Claim, Retrieve Add. Data, Check Claim, Reject Claim, Call Costumer.
3. Ablauf
   Record Claim, Check Claim, Retrieve Add. Data, Check Claim, Settle Claim, Send Letter, Calculate Loss Amount, Recalc. Cost. Contr., Payout.

Block C: UML/P Aktivitätsdiagramme

Frage C.1: Wie gut hat Ihnen die Angabe des syntaktischen Unterschieds beim Verstehen der Unterschiede der Modelle geholfen?

☐	☐	☐	☐
Sehr gut	eher gut	eher wenig	sehr wenig

Frage C.2: Wie gut hat Ihnen die Angabe des semantischen Unterschieds beim Verstehen der Unterschiede der Modelle geholfen?

☐	☐	☐	☐
Sehr gut	eher gut	eher wenig	sehr wenig

Frage C.3: Wie gut hat Ihnen die Erläuterung der Unterschiede durch eine Kombination der syntaktischen mit den semantischen Unterschieden beim Verstehen der Unterschiede der Modelle geholfen?

☐	☐	☐	☐
Sehr gut	eher gut	eher wenig	sehr wenig

Frage C.4: Wie gut hat Ihnen die Beispielabstraktion beim Verstehen der Unterschiede der Modelle geholfen?

☐	☐	☐	☐
Sehr gut	eher gut	eher wenig	sehr wenig

Frage C.5: Wie gut hat Ihnen die Zusammenfassung der semantischen Unterschiede beim Verstehen der Unterschiede der Modelle geholfen?

☐	☐	☐	☐
Sehr gut	eher gut	eher wenig	sehr wenig

Unterschiede zwischen State Charts

Im Folgenden wird ein Scenario zur Evolutionsanalyse von State Charts vorgestellt. Im Scenario werden zwei State Charts gezeigt. Das mit v2 gekennzeichnete State Chart ist die Nachfolgerversion des mit v1 gekennzeichneten State Charts. Es werden der syntaktische Unterschied und der semantische Unterschied des State Charts mit der Version v2 zum State Chart mit der Version v1 erläutert.

Versuchen Sie bitte die erläuterten Unterschiede nachzuvollziehen. Bewerten Sie bitte jeweils anschließend auf dieser Grundlage inwiefern Ihnen die Erläuterungen beim Verständnis der Unterschiede von der Nachfolgerversion v2 zur Version v1 geholfen haben.

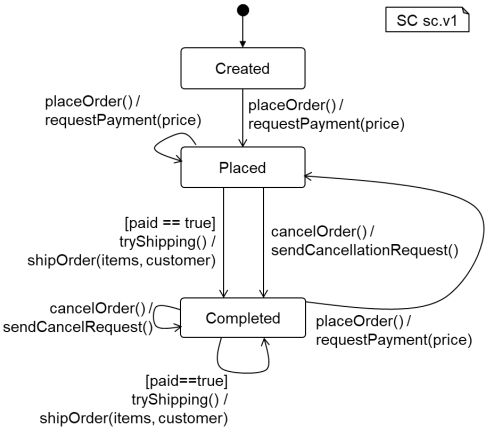
Block D: State Charts

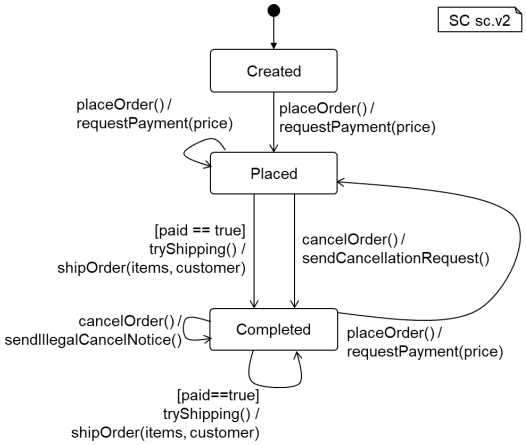
State Chart Scenario

Stellen Sie sich vor, Sie sind Entwickler eines Online-Shops. Der Kunde soll durch Befüllen eines virtuellen Einkaufskorbes die Artikel der Bestellung angeben können. Hat der Kunde dem Warenkorb alle gewünschten Artikel hinzugefügt und dies über einen Button-Klick bestätigt, soll der Kunde aus mehreren Zahlungsmethoden wählen oder den Abschluss der Bestellung vertagen können. Das System soll sicherstellen, dass nur gestellte und auch bezahlte Bestellungen in den Versand gehen können. Solange Bestellungen gestellt, aber noch nicht versandt sind, sollen die Kunden die Möglichkeit haben, die Bestellung zu stornieren. Gegebenenfalls können stornierte Bestellungen wiederaufgenommen und bezahlt werden, ohne dass der Einkaufskorb erneut gefüllt werden muss. Bezahlte Bestellungen müssen immer versandt werden. Sie modellieren das unten gezeigte State Chart sc.v1.

Dem Besitzer des Online-Shops fällt nun noch eine weitere Anforderung ein.

Ein Arbeitskollege ändert daraufhin das State Chart sc.v1. Daraus entsteht das State Chart sc.v2. Sie möchten die Unterschiede vom neuen State Chart sc.v2 zum State Chart sc.v1 verstehen.





Block D: State Charts

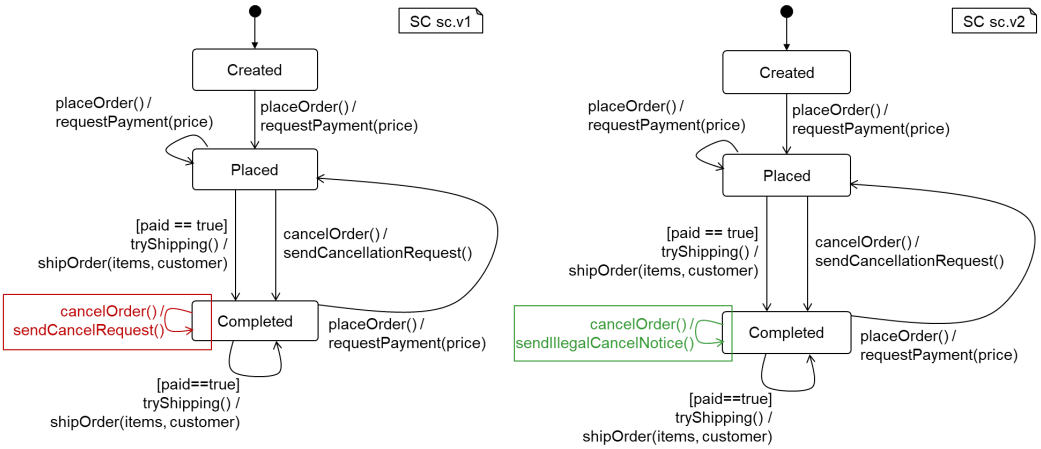
Syntaktische Unterschiede im State Chart Scenario (D1)

Erläuterung des syntaktischen Unterschieds von sc1.v2 zu sc1.v1 in textueller Form:

1. Die Transition vom Zustand Completed zum Zustand Completed mit dem Stimulus cancelOrder() und der Aktion sendIllegalCancelNotice() wurde hinzugefügt.
2. Die Transition vom Zustand Completed zum Zustand Completed mit dem Stimulus cancelOrder() und der Aktion sendCancelRequest() wurde gelöscht.

Erläuterung des syntaktischen Unterschieds in grafischer Form:

Die im folgenden Bild im State Chart sc1.v1 rot dargestellten und durch einen Kasten markierten Elemente wurden gelöscht oder geändert. Die im State Chart sc2.v2 grün dargestellten und durch einen Kasten markierten Elemente wurden hinzugefügt oder geändert.



Semantische Unterschiede im State Chart Scenario

Erläuterung des semantischen Unterschieds in textueller Form (D2):

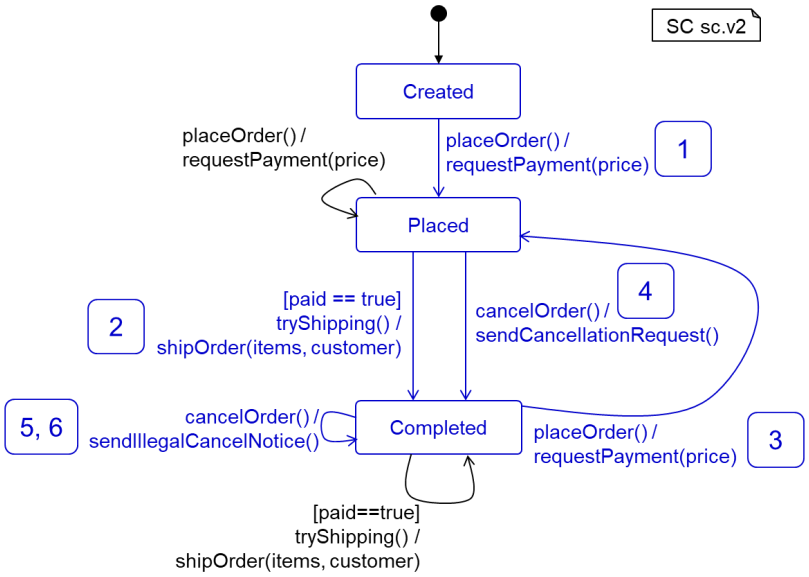
Die folgenden Abläufe sind im Statechart sc1.v2 möglich und im Statechart sc1.v1 nicht möglich:

1. Ablauf
   (Created, placeOrder() / requestPayment(price), Placed),
   (Placed, tryShipping() / shipOrder(items, customer), Completed),
   (Completed, placeOrder() / requestPayment(price), Placed),
   (Placed, cancelOrder() / sendCancellationRequest(), Completed),
   (Completed, cancelOrder() / SendIllegalCancelNotice(), Completed),
   (Completed, cancelOrder() / SendIllegalCancelNotice(), Completed)
2. Ablauf:
   (Created, placeOrder() / requestPayment(price), Placed),
   (Placed, tryShipping() / shipOrder(items, customer), Completed),
   (Completed, cancelOrder() / SendIllegalCancelNotice(), Completed)
3. Ablauf
   (Created, placeOrder() / requestPayment(price), Placed),
   (Placed, placeOrder() / requestPayment(price), Placed),
   (Placed, tryShipping() / shipOrder(items, customer), Completed),
   (Completed, cancelOrder() / SendIllegalCancelNotice(), Completed),
   (Completed, tryShipping() / shipOrder(items, customer), Completed)

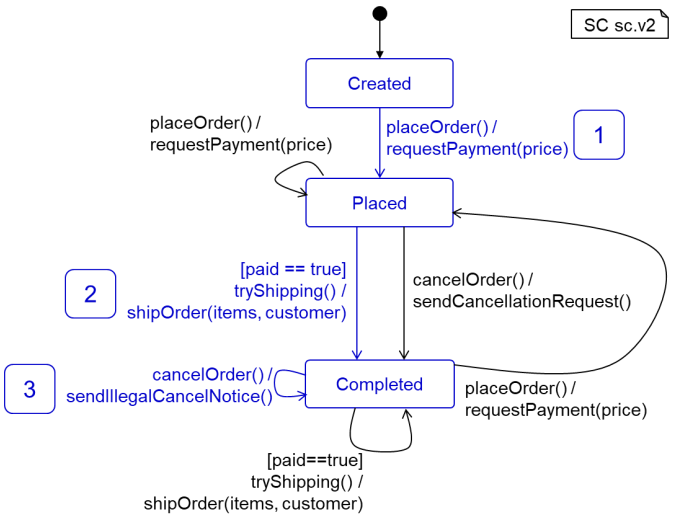
Erläuterung des semantischen Unterschieds in grafischer Form (D2):

In den folgenden drei Bildern werden die zuvor erläuterten Abläufe 1, 2 und 3 im State Chart sc.v2 grafisch dargestellt:

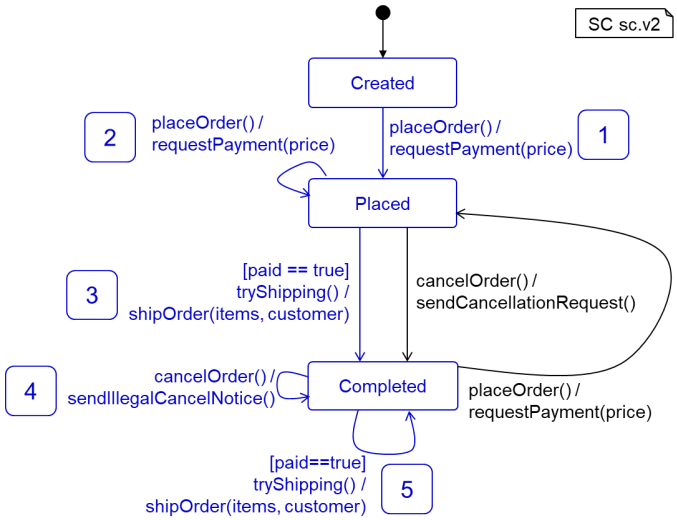
1. Ablauf:



2. Ablauf:



3. Ablauf:



Erläuterung des Unterschieds durch eine Kombination der syntaktischen Unterschiede mit den semantischen Unterschieden (D3):

Durch die Anwendung der folgenden Änderungen am State Chart sc1.v2 erhält man ein State Chart, das ausschließlich Abläufe modelliert, die auch im State Chart sc1.v1 modelliert sind:

1. Die Transition vom Zustand Completed zum Zustand Completed mit dem Stimulus cancelOrder() und der Aktion sendIllegalCancelNotice() löschen.

Erläuterungen des semantischen Unterschieds durch Beispielabstraktion (D4):

1. Wenn die Events sendIllegalCancelNotice() und sendCancelRequest() in den Aktionen beider Statecharts gelöscht werden würden, dann wäre jeder Ablauf von sc1.v2 auch ein Ablauf von sc1.v1.
2. Wenn das Event sendIllegalCancelNotice() im State Chart sc1.v2 durch das Event sendCancelRequest() ersetzt werden würde, dann wäre jeder Ablauf von sc1.v2 auch ein Ablauf von sc1.v1.

Erläuterung des semantischen Unterschieds durch Zusammenfassung (D5):

Die folgende Liste repräsentiert eine Menge von Abläufen, die in sc1.v2 möglich und im State Chart sc1.v1 nicht möglich sind. Die Mengen der Stimulus/Reaktion Paare aller Abläufe unterscheiden sich paarweise.

1. Ablauf
   (Created, placeOrder() / requestPayment(price), Placed),
   (Placed, tryShipping() / shipOrder(items, customer), Completed),
   (Completed, cancelOrder() / SendIllegalCancelNotice(), Completed)
2. Ablauf
   (Created, placeOrder() / requestPayment(price), Placed),
   (Placed, tryShipping() / shipOrder(items, customer), Completed),
   (Completed, placeOrder() / requestPayment(price), Placed),
   (Placed, cancelOrder() / sendCancellationRequest(), Completed),
   (Completed, cancelOrder() / SendIllegalCancelNotice(), Completed),
   (Completed, cancelOrder() / SendIllegalCancelNotice(), Completed)

Block D: State Charts

Frage D.1: Wie gut hat Ihnen die Angabe des syntaktischen Unterschieds beim Verstehen der Unterschiede der Modelle geholfen?

☐ Sehr gut ☐ eher gut ☐ eher wenig ☐ sehr wenig

Frage D.2: Wie gut hat Ihnen die Angabe des semantischen Unterschieds beim Verstehen der Unterschiede der Modelle geholfen?

☐ Sehr gut ☐ eher gut ☐ eher wenig ☐ sehr wenig

Frage D.3: Wie gut hat Ihnen die Erläuterung der Unterschiede durch eine Kombination der syntaktischen mit den semantischen Unterschieden beim Verstehen der Unterschiede der Modelle geholfen?

☐ Sehr gut ☐ eher gut ☐ eher wenig ☐ sehr wenig

Frage D.4: Wie gut haben Ihnen die Beispielabstraktionen beim Verstehen der Unterschiede der Modelle geholfen?

☐ Sehr gut ☐ eher gut ☐ eher wenig ☐ sehr wenig

Frage D.5: Wie gut hat Ihnen die Zusammenfassung der semantischen Unterschiede beim Verstehen der Unterschiede der Modelle geholfen?

☐ Sehr gut ☐ eher gut ☐ eher wenig ☐ sehr wenig

Unterschiede zwischen Feature Diagrammen

Im Folgenden wird ein Scenario zur Evolutionsanalyse von Feature Diagrammen vorgestellt. Im Scenario werden zwei Feature Diagramme gezeigt. Das mit v2 gekennzeichnete Feature Diagramm ist die Nachfolgerversion des mit v1 gekennzeichneten Feature Diagramms. Es werden der syntaktische Unterschied und der semantische Unterschied des Feature Diagramms mit der Version v2 zum Feature Diagramm mit der Version v1 erläutert.

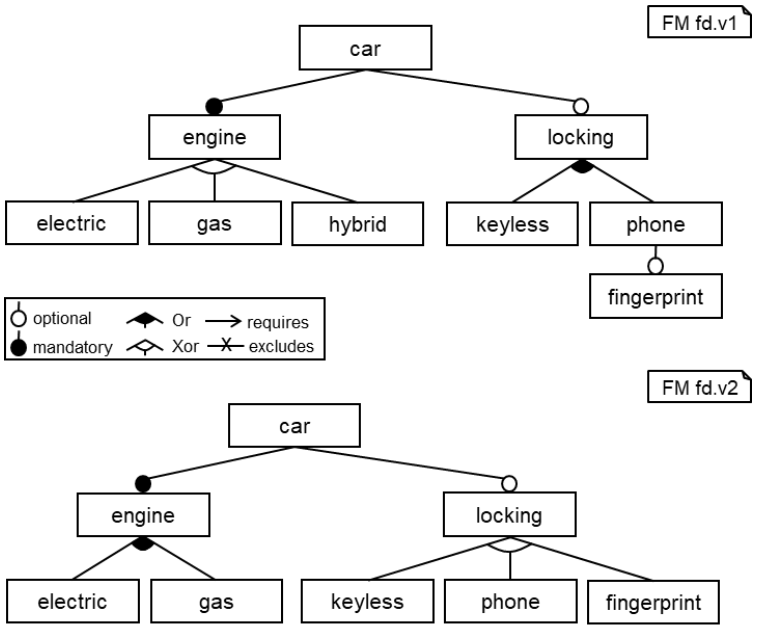
Versuchen Sie bitte die erläuterten Unterschiede nachzuvollziehen. Bewerten Sie bitte jeweils anschließend auf dieser Grundlage inwiefern Ihnen die Erläuterungen beim Verständnis der Unterschiede von der Nachfolgerversion v2 zur Version v1 geholfen haben.

Block E: Feature Diagramme

Feature Diagramm Scenario

In diesem Beispiel sind Sie Entwickler einer Automobil-Produktlinie. Der Hersteller entwickelt Fahrzeuge mit Elektrischem-, Gas- oder Hybrid-Antrieb. Das Alleinstellungsmerkmal des Herstellers sind die innovativen Verriegelungen, die der Kunde auf Wunsch erwerben kann. Zum einen bietet der Hersteller eine schlüssellose Verriegelung an. Die andere Möglichkeit ist eine Verriegelung, die per Smartphone und optional sogar über den im Smartphone integrierten Fingerabdruck-Sensor bedient werden kann. Wahlweise kann der Kunde beide Verriegelungssysteme in sein Fahrzeug verbauen lassen. Sie modellieren dazu das Feature Diagramm fd.v1.

Es hat sich nach Absprache mit den Händler eine Änderung ergegeben. Ein Arbeitskollege ändert daraufhin das Feature Diagramm fd.v1. Daraus entsteht das Feature Diagramm fd.v2. Sie möchten die Unterschiede vom neuen Feature Diagramm fd.v2 zum Feature Diagramm fd.v1 verstehen.



Block E: Feature Diagramme

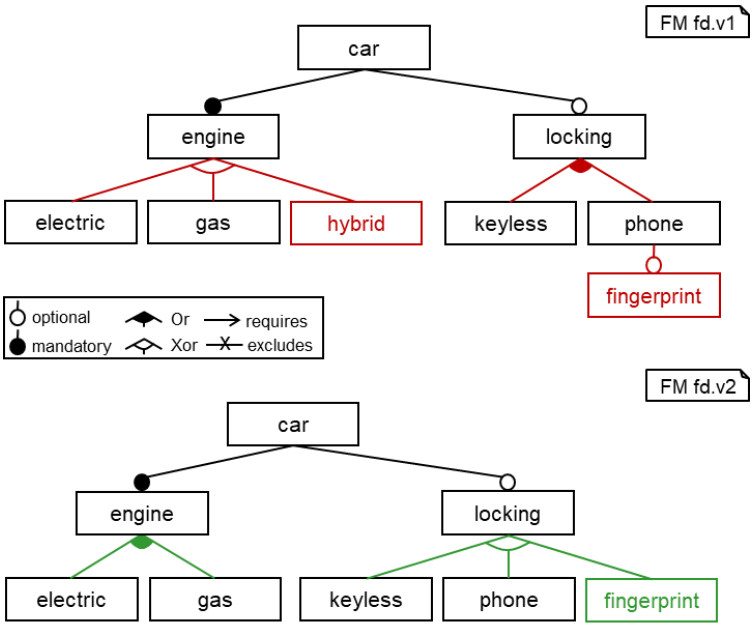
Syntaktische Unterschiede im Feature Diagramm Scenario (E1)

Erläuterung des syntaktischen Unterschieds von fd.v2 zu fd.v1 in textueller Form:

1. Die Gruppe des Features engine mit den Features electric, gas und hybrid wurde zu einer Or-Gruppe konvertiert.
2. Das Feature hybrid wurde von der Gruppe des Features engine mit den Features electric, gas und hybrid entfernt.
3. Das Feature fingerprint wurde entfernt.
4. Die Gruppe des Features locking mit den Features keyless und phone wurde zu einer Xor-Gruppe konvertiert.
5. Der Gruppe des Features locking mit den Features keyless und phone wurde das Feature fingerprint hinzugefügt.

Erläuterung des syntaktischen Unterschieds in grafischer Form:

Die im folgenden Bild im Feature Diagramm fd.v1 rot dargestellten und durch einen Kasten markierten Elemente wurden gelöscht oder geändert. Die im Feature Diagramm fd.v2 grün dargestellten und durch einen Kasten markierten Elemente wurden hinzugefügt oder geändert.



Semantische Unterschiede im Klassendiagramm Scenario

Erläuterung des semantischen Unterschieds in textueller Form (E2):

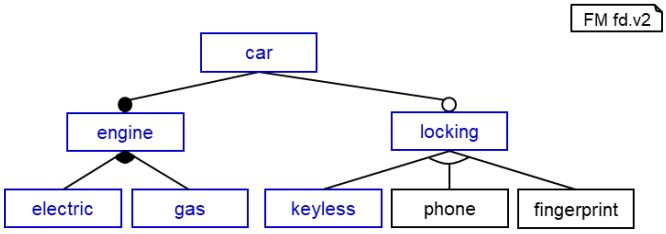
Die folgenden Konfigurationen sind im Feature Diagramm fd.v2 valide und im Feature Diagramm fd.v1 nicht valide:

1. Konfiguration:
   {car, engine, electric, gas, locking, keyless}
2. Konfiguration:
   {car, engine, electric, gas, locking, phone}
3. Konfiguration:
   {car, engine, electric, gas}
4. Konfiguration:
   {car, engine, electric, gas, locking, fingerprint}
5. Konfiguration:
   {car, engine, electric, gas, locking, phone}
6. Konfiguration:
   {car, engine, gas, locking, fingerprint}

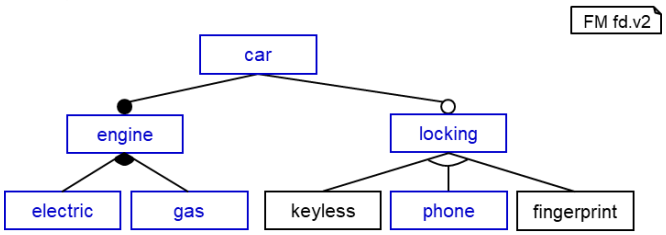
Erläuterung des semantischen Unterschieds in grafischer Form:

Im den folgenden sechs Bildern werden die zuvor gezeigten Konfigurationen 1, 2, 3, 4, 5 und 6 im Feature Diagramm fd.v2 grafisch dargestellt:

1. Konfiguration:

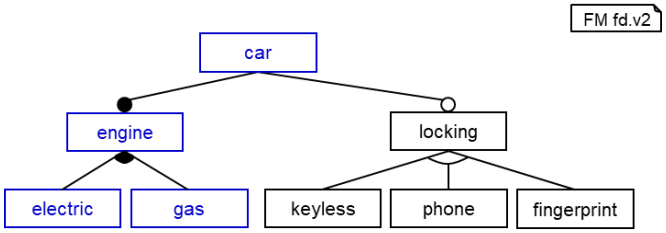


2. Konfiguration:

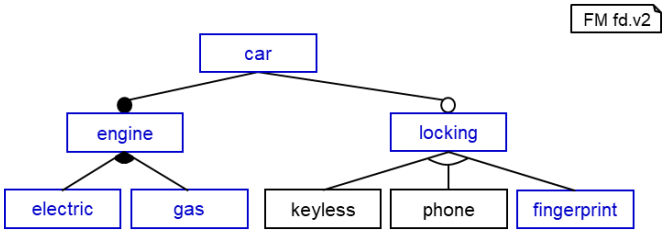


Block E: Feature Diagramme

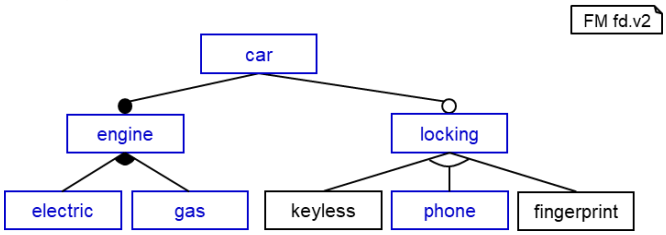
3. Konfiguration:



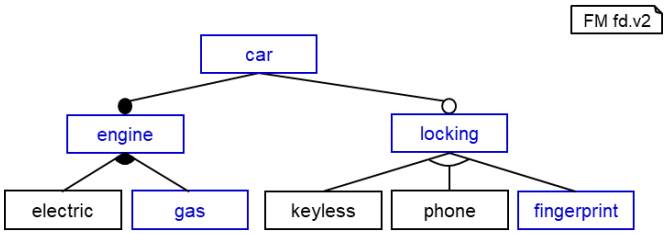
4. Konfiguration:



5. Konfiguration:



6. Konfiguration:



Erläuterung des Unterschieds durch eine Kombination der syntaktischen Unterschiede mit den semantischen Unterschieden (E3):

Wenn die folgenden Änderungen am Feature Diagramm fd.v2 durchgeführt werden, dann erhält man ein Feature Diagramm, das ausschließlich valide Konfigurationen hat, die auch valide Konfigurationen des Feature Diagramms fd.v1 sind:

1. Das Feature fingerprint entfernen.
2. Die Gruppe des Features engine mit den Features electric und gas zu einer Xor-Gruppe konvertieren.

Erläuterungen des semantischen Unterschieds durch Beispielabstraktion (E4):

1. Wenn man von der Existenz der Features fingerprint und gas in den Feature Diagrammen abstrahieren würde, dann wäre jede valide Konfiguration von fd.v2 auch eine valide Konfiguration von fd.v1.
2. Wenn man von der Existenz der Features fingerprint und electric in den Feature Diagrammen abstrahieren würde, dann wäre jede valide Konfiguration von fd.v2 auch eine valide Konfiguration von fd.v1.

Erläuterung des semantischen Unterschieds durch Zusammenfassung (E5):

Die folgende Liste von Konfigurationen repräsentiert eine Menge von validen Konfigurationen von fd.v2, die keine validen Konfigurationen von fd.v1 sind. Die Anzahl der Elemente der Konfigurationen unterscheiden sich paarweise.

1. Konfiguration:
   {car, engine, electric, gas}
2. Konfiguration:
   {car, engine, electric, gas, locking, fingerprint}

Block E: Feature Diagramme

Frage E.1: Wie gut hat Ihnen die Angabe des syntaktischen Unterschieds beim Verstehen der Unterschiede der Modelle geholfen?

☐ Sehr gut ☐ eher gut ☐ eher wenig ☐ sehr wenig

Frage E.2: Wie gut hat Ihnen die Angabe des semantischen Unterschieds beim Verstehen der Unterschiede der Modelle geholfen?

☐ Sehr gut ☐ eher gut ☐ eher wenig ☐ sehr wenig

Frage E.3: Wie gut hat Ihnen die Erläuterung der Unterschiede durch eine Kombination der syntaktischen mit den semantischen Unterschieden beim Verstehen der Unterschiede der Modelle geholfen?

☐ Sehr gut ☐ eher gut ☐ eher wenig ☐ sehr wenig

Frage E.4: Wie gut haben Ihnen die Beispielabstraktionen beim Verstehen der Unterschiede der Modelle geholfen?

☐ Sehr gut ☐ eher gut ☐ eher wenig ☐ sehr wenig

Frage E.5: Wie gut hat Ihnen die Zusammenfassung der semantischen Unterschiede beim Verstehen der Unterschiede der Modelle geholfen?

☐ Sehr gut ☐ eher gut ☐ eher wenig ☐ sehr wenig

Block F: Abschluss

Sie haben jetzt unterschiedliche Möglichkeiten zur Identifikation von syntaktischen und semantischen Unterschieden zwischen Modellen in vier unterschiedlichen Modellierungssprachen kennengelernt.

Frage F.1: Welche Anwendungsfälle können sie sich für diese Möglichkeiten der Identifikation von syntaktischen und semantischen Unterschieden vorstellen?

Vielen Dank für die Beantwortung!

Aachener Informatik-Berichte

This list contains all technical reports published during the past three years. A complete list of reports dating back to 1987 is available from:

`http://aib.informatik.rwth-aachen.de/`

To obtain copies please consult the above URL or send your request to:

Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen,
Email: `biblio@informatik.rwth-aachen.de`

2017-01 *	Fachgruppe Informatik: Annual Report 2017
2017-02	Florian Frohn and Jürgen Giesl: Analyzing Runtime Complexity via Innermost Runtime Complexity
2017-04	Florian Frohn and Jürgen Giesl: Complexity Analysis for Java with AProVE
2017-05	Matthias Naaf, Florian Frohn, Marc Brockschmidt, Carsten Fuhs, and Jürgen Giesl: Complexity Analysis for Term Rewriting by Integer Transition Systems
2017-06	Oliver Kautz, Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe: CD2Alloy: A Translation of Class Diagrams to Alloy
2017-07	Klaus Leppkes, Johannes Lotz, Uwe Naumann, and Jacques du Toit: Meta Adjoint Programming in C++
2017-08	Thomas Gerlitz: Incremental Integration and Static Analysis of Model-Based Automotive Software Artifacts
2017-09	Muhammad Hamad Alizai, Jan Beutel, Jó Ágila Bitsch, Olaf Landsiedel, Luca Mottola, Przemyslaw Pawelczak, Klaus Wehrle, and Kasim Sinan Yildirim: Proc. IDEA League Doctoral School on Transiently Powered Computing
2018-01 *	Fachgruppe Informatik: Annual Report 2018
2018-02	Jens Deussen, Viktor Mosenkis, and Uwe Naumann: Ansatz zur variantenreichen und modellbasierten Entwicklung von eingebetteten Systemen unter Berücksichtigung regelungs- und softwaretechnischer Anforderungen
2018-03	Igor Kalkov: A Real-time Capable, Open-Source-based Platform for Off-the-Shelf Embedded Devices
2018-04	Andreas Ganser: Operation-Based Model Recommenders
2018-05	Matthias Terber: Real-World Deployment and Evaluation of Synchronous Programming in Reactive Embedded Systems
2018-06	Christian Hensel: The Probabilistic Model Checker Storm - Symbolic Methods for Probabilistic Model Checking
2019-01 *	Fachgruppe Informatik: Annual Report 2019
2019-02	Tim Felix Lange: IC3 Software Model Checking
2019-03	Sebastian Patrick Grobosch: Formale Methoden für die Entwicklung von eingebetteter Software in kleinen und mittleren Unternehmen
2019-05	Florian Göbe: Runtime Supervision of PLC Programs Using Discrete-Event Systems

2020-01 * Fachgruppe Informatik: Annual Report 2020

2020-02 Towards an Isabelle Theory for distributed, interactive systems - the untimed case

2020-03 John F. Schommer:

* These reports are only available as a printed version.
Please contact biblio@informatik.rwth-aachen.de to obtain copies.