Sometimes You Have to Treat the Symptoms: Tackling Model Drift in an Industrial Clone-and-Own Software Product Line

Christof Tinnes
Siemens AG – Technology, München, Germany

Wolfgang Rössler
Siemens Mobility GmbH, München, Germany

Uwe Hohenstein
Siemens AG – Technology, München, Germany

Torsten Kühn
Siemens Mobility GmbH, München, Germany

Andreas Biesdorf
Siemens AG – Technology, München, Germany

Sven Apel
Saarland University, Informatics Campus, Saarbrücken, Germany

ABSTRACT

Many industrial software product lines use a clone-and-own approach for reuse among software products. As a result, the different products in the product line may *drift apart*, which implies increased efforts for tasks such as change propagation, domain analysis, and quality assurance. While many solutions have been proposed in the literature, these are often difficult to apply in a real-world setting. We study this drift of products in a concrete large-scale industrial model-driven clone-and-own software product line in the railway domain at our industry partner. For this purpose, we conducted interviews and a survey, and we investigated the models in the model history of this project. We found that increased efforts are mainly caused by large model differences and increased communication efforts. We argue that, in the short-term, treating the symptoms (i.e., handling large model differences) can help to keep efforts for software product-line engineering acceptable — instead of employing sophisticated variability management. To treat the symptoms, we employ a solution based on semantic-lifting to simplify model differences. Using the interviews and the survey, we evaluate the feasibility of variability management approaches and the semantic-lifting approach in the context of this project.

CCS CONCEPTS

• **Software and its engineering → Software development methods**; **Software product lines**; **Software configuration management and version control systems**.

KEYWORDS

model-driven engineering, software evolution, software product lines

ACM Reference Format:
Christof Tinnes, Wolfgang Rössler, Uwe Hohenstein, Torsten Kühn, Andreas Biesdorf, and Sven Apel. 2022. Sometimes You Have to Treat the Symptoms: Tackling Model Drift in an Industrial Clone-and-Own Software Product Line. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3540250.3558960

1 INTRODUCTION

Software product-line engineering [41] aims at developing a family of software products by reusing software artifacts as well as processes across the family. There are several software product-line engineering methodologies [2, 44], ranging from simple cloning of artifacts across products (clone-and-own) to platforms with built-in variability support, which allow to generate products based on a configuration (i.e., a feature selection). Depending on various factors (e.g., the number of products), it makes sense to favor one approach over another [14, 30, 41, 44]. For example, the overhead for establishing and using a platform will only pay off when the platform is used for a larger number of products [41]. It is important to note that the reuse of artifacts in software product lines is not limited to source code, but can also include other artifacts, such as documentation, requirements, and models [3, 4, 7, 35, 39, 55].

Many industrial software product lines have been grown organically, using some form of managed cloning (clone-and-own with some form of clone management in place) [14, 44]. As a consequence, changes have to be propagated from one product to another. This is often done manually and therefore time-consuming and error-prone [30]. A key challenge of the evolution of clone-and-own software product lines is that products tend to drift apart [28, 58]. This leads to a risk of losing the benefits of reuse if efforts related to cloning and merging exceed the savings of reuse.

In this experience paper, we report on a model-driven software product line (i.e., models are the primary artifacts) in the railway domain at our industry partner, which employs a form of managed cloning. In particular, we study the divergence of models in this setting. We observed that quality assurance, change propagation, and domain analysis are the main drivers for increased effort (or costs), due to the cloning approach — an observation consistent with the literature [14, 30, 45]. Nevertheless, these activities are indispensable. Especially for safety-relevant parts of the system, a thorough quality assurance of the changes made to the system is critical to decrease the risk of failing software qualification, which in turn leads to an increase in time-to-market.

Common to all of these activities is the analysis of *differences* between products or between a product and the common platform. In large software product lines, the models and also the differences between models easily become huge. We found that, indeed, large model differences are a major symptom of the drift at our industry partner. It is increasingly difficult to identify commonalities that can be lifted to a common platform or propagated across products. Many small but uncritical changes, such as model element renamings, make model differences unnecessarily verbose and confusing. For example, in our case study, renaming a package leads to the change of the fully qualified name of subordinated elements and therefore to plenty of fine-grained changes in the model difference.

A solution to the divergence problem might be to migrate to a platform-based approach [14, 30, 44]. This is not only a time-consuming task, but there is also a risk of losing flexibility to create new products. Moreover, the independence of products in a clone-and-own software product line is sometimes desired to avoid unintended side effects [14, 30]. In general, many of the assumptions made by platform approaches are often not compatible with real-world constraints such as organizational structures within a company. An alternative to platform-based approaches is to utilize feature traces in a bottom-up manner for change propagation and the composition of new products [18, 28]. Unfortunately, state-of-the-art tools are not mature yet (e.g., scalability to large product lines has not been shown yet) and are not available for model artifacts.

While it is desirable in the long term to systematically manage variability, it is imperative in the short term to "treat the symptoms" of a clone-and-own approach. That is, we need to reduce the effort required for critical activities such as quality assurance, change propagation, and domain analysis. For this purpose, we have combined semantic lifting [27] with a recent approach for edit operation mining [54], to effectively simplify large model differences at our industry partner. The idea of this semantic lifting approach is to *compress* the differences, that is, to combine many fine-grained differences into higher-level, reusable change patterns.

Based on semi-structured interviews and a questionnaire survey, we explore the merits and barriers of platform-based approaches, bottom-up approaches, and the semantic lifting approach in the context of drift between products in an industrial setting.

In a summary, we make the following contributions:

- We report on a case study and challenges related to model drift in a real-world industrial clone-and-own software product line in the railway domain with more than 200 engineers, 300 GB of artifacts, and over 7 years of history.
- We conducted interviews and a survey with practitioners in this project to
  - verify the presence of model drift in this real-world case study,
  - study symptoms and causes for the model drift,
  - evaluate whether approaches proposed in the literature can be applied and whether a migration would be feasible,
  - evaluate a bottom-up approach based on semantic lifting and pattern mining to mitigate one of the major symptoms of model drift, that is, large model differences.
- We find that major symptoms of model drift are large model differences as well as increased communication efforts, mainly for tasks such as quality assurance, change propagation, and domain analysis. There are several causes, most prominently an increasing model size as well as missing tool support for several variability-related tasks. Worse, available approaches cannot be applied because the cost and risk of a migration would be to high, existing tools are not mature, and critical features are missing. A semantic lifting approach can help to mitigate challenges related to large model differences.

2 STATE OF THE ART

Supporting and managing model evolution, for example, supporting collaborative work on models, versioning of models, or quality assurance of the models has been studied extensively in the literature [1, 6, 26, 29, 34, 37, 43, 53]. The studies by Straeten et al. [53], Kahlil et al. [29], and Paige et al. [37] provide an extensive overview.

Whereas model evolution research is focusing on the evolution in the *temporal dimension*, the software product-line engineering community is interested in differences in the *product dimension.* In the literature, several alternatives and approaches complementary to clone-and-own are proposed. In what follows, we will recapture the clone-and-own approach, describe the problem of the so called *unintentional divergence* in clone-and-own, and describe possible solutions proposed in the literature. Furthermore, we provide an overview of industrial clone-and-own case studies.

2.1 Clone-and-Own

Clone-and-own [14, 28, 30, 44, 45] describes the *ad-hoc* process of cloning existing products and developing the cloned products independently from each other. There has been a debate in the software product-line engineering community about whether a clone-and-own approach should be discouraged or not [14, 24, 28]. Nevertheless, as a matter of fact [14, 28, 44], clone-and-own approaches are used in the industrial practice. In many cases utilizing clone-and-own approaches, there exists a single product which has to be modified slightly to satisfy the needs of other customers. Cloning is then an available mechanism, which can be rapidly used to satisfy this customers needs. Often it is not known in advance whether an existing product will evolve into a family of products. Therefore, it is hard to assess whether the initial effort to setup a platform for the product family will pay off in the long run [41]. An observation made by Berger et al. [8] is "that none of our subjects exercises pure clone&own, but that variants already use variation points". This observation is also true for our railway case study, where trainset platforms also have some internal variability (see Section 3).

A common issue related to the clone-and-own approach is the unintentional divergence between the products [28, 48]. *Unintentional divergence* describes the increasing difference between products in the product family. There are multiple possible reasons for this divergence, for example, missing reuse opportunities (e.g., commonalities are not identified during domain analysis), *noise* due to differences in the implementation of features in different products, or even the increased complexity of the individual products as a consequence of the evolution of the products. In the context of a model-driven product line, we will refer to *model drift* as a specific instance of unintentional divergence in model artifacts.

2.2 Platform Approaches

One solution to the divergence problem is to migrate the cloned product line to a *150% platform* [14, 30, 44]. There are only a few tools that support 150% platform approaches for model-driven software product lines. SuperMod [49] relies on so-called *filtered editing* and *feature ambitions*, which a user can select when committing changes. The MagicDraw Product Line Engineering Plugin is more mature, but the user has to explicitly model the variability for every (variable) model element. In contrast to 150% platform approaches, where a product is generated by selecting a subset of the functionality implemented in the 150% platform, compositional approaches [3, 17, 22, 40, 46] have been proposed in the literature. For example, Apel et al. [3] propose an approach to compose a specific product by superimposing *model fragments*; typically one model fragment per feature. Jayaraman et al. [22] describe a UML composition language based on graph transformations for reusing features. Another compositional approach is *delta modeling* [46], where a product is composed out of a *core model* (a valid product) and multiple delta-models that encapsulate modifications to the core model. Delta-models are not necessarily in one-to-one correspondence with features. Elrad et al. [17] propose an approach for aspect-oriented modelling to separate crosscutting-concerns and core functionality in UML models.

A migration from a clone-and-own approach to a platform approach is not only time-consuming, but there is also a risk of losing flexibility to create new products, and the independence of the products in a clone-and-own software product line is sometimes desired to avoid unintended side effects [14, 30].

2.3 Bottom-up Product Line Engineering

Some authors observe that, with an increasing number of products, maintenance efforts for a clone-and-own approach will rapidly grow [14, 28, 38, 57], but they also acknowledge that, in some settings (e.g., a small number of products), a cloning approach is viable [24, 28, 38, 41]. Furthermore, many industrial product lines start with a single product, and it is not clear at the beginning, how many products there will be in the future or how long the product line will be used at all [14, 27].

For this reason, approaches supporting variability management in clone-and-own product lines have been proposed [18, 28]. The basic idea is to utilize feature traces in a bottom-up manner for change propagation and the composition of new products. Some authors propose *variation control systems* [18, 25, 31, 49, 51], which aim at unifying version control and variability management[1].

2.4 Industrial Clone-and-Own Case Studies

Some authors study clone-and-own in multiple-case studies (i.e., having more than one concrete subject project). For example, Rubin et al. [44] evaluate a framework for managing collections of related products based on three industrial case studies. Dubinsky et al. [14] study cloning practices in six industrial clone-and-own software product lines. More recently, Krüger et al. [30] compare cost and cost factors of clone-and-own and platform-oriented reuse in industrial settings using a literature review and interviews within one company. Berger et al. [8] study the state of adoption of a systematic variability management in twelve industry case studies across several domains. While this research compares platform-oriented research and clone-and-own or investigates a transition from clone-and-own to a more systematic variability management, we study the concrete challenge of drift between products in one concrete industrial clone-and-own software product line.

There are also single-case studies in an industrial context, for example, Jepsen et al. [23] or Weston et al. [56] focusing on the extractive adoption of software product lines. Other single-case studies compare platform-based and clone-and-own approaches, for example, Echeverría et al. [15] compare effectiveness, efficiency, and satisfaction of clone-and-own and a platform-based approach in a controlled experiment. A comprehensive list of case studies using extractive adoption is given by the *ESPLA catalog* [32]. To the best of our knowledge, there is no work studying drift between products in an industrial setting.

3 RAILWAY CASE STUDY SETTING

In this section, we describe the setting of our case study that we conducted with our industry partner.

Our industry partner offers a wide range of trains from small commuter rails to high-speed trains. Their trains include a lot of software components that provide various functionality, from heating, ventilation, and air conditioning (HVAC) systems, to highly safety-relevant systems such as the drive control system. The software components consist mainly of SCL code[2]. Over the years, our industry partner's code bases have become large and complex. For this reason, the engineering team that is responsible for the train software decided in 2014 to follow a *model-driven engineering* (MDE) methodology. This eases the documentation of the train software and maintains traceability between requirements and software components. The team of around 200 engineers uses MagicDraw[21] as their modelling environment, because it is an industry-proven tool, supports modelling in the modelling language used in this project (i.e., SysML), and is capable of handling large models. As a further benefit of MDE, large parts of the software components' source code are generated out of the SysML models. For code generation, MagicDraw models are transformed into Eclipse Modeling Framework (EMF) models [52], which are then transformed into SCL source code components. The reason to use EMF in an intermediate step is the availability of tooling and libraries from the EMF ecosystem. Code generation is a one-way process, that is, no round-trip engineering is performed.

At the time of the study, the model base consisted of approximately 300GB of artifacts (including image data for diagrams). The overall model for the entire train software is divided into 122 weakly coupled submodels, which are evolved and versioned independently from each other. For instance, there is one submodel for HVAC and another submodel for drive and break control.

The train domain is a highly regulated domain, so the software is subject to qualification and certification requirements according to IEC 61508 [20], EN 50126 [10], EN 50657 [12], and EN 50129 [11]. Maintaining requirements traceability and exhaustive documentation is therefore required, and following a model-driven approach helps to automate many of the documentation related tasks. In Table 1, we give an overview of the project setting.

[1]Some variation control systems internally maintain 150% model, and therefore belong to the category of platform approaches.

[2]Aka. structured text; a block-structured IEC 61131-3 language, mainly designed for programmable logic controllers.

Since many of the software components shall be reused across different trainsets such as commuter rail, light rail, and high-speed trains, the software is developed as a software product line. The artifacts are versioned in a repository with one branch per trainset, called *trainset platform*. There is also a platform, called the *common platform*, which holds the type-independent parts of the models and the ones that can be configured/adjusted in the train-specific branches. The common platform is evolved by following an extractive software product line adoption path [2], that is, newly developed features and reusable parts are (manually) identified and regularly merged into the common platform (see Figure 1). Common functionality is evolved in the common platform and propagated to trainset platforms, if desired. Furthermore, fixes or changes to the models for the trainset types have to be propagated to the common platform and trainset types that use the affected parts.

In general, *change and feature propagation* has to be performed carefully, since parts of the software system might have been qualified already and must not be modified thereafter. Several times per month, for *quality assurance* purposes, the common platform and trainset platform branches are compared in the modelling tool MagicDraw and manually checked. Incompatible or incomplete changes have to be be identified during these checks. Some changes might even cause serious problems if merged into the common platform and therefore need to be identified and fixed. Furthermore, based on the model differences, a *domain analysis* is performed by a domain expert to identify commonalities and reuse opportunities.

Table 1: Project context for our railway case study. The model size is given in number of model elements (attributes not included). Model differences are between a trainset platform and the common platform and include changed attributes.

Number of developers	Train types	Sub-models	Avg. model size	Avg. number of differences	Artifact size	Age
200	8	122	24.627	63.096	300 GB	7 yrs

4 SEMANTIC LIFTING TO REDUCE THE COMPLEXITY OF MODEL DIFFERENCES

As a consequence of unintentional divergence, large model differences (e.g., thousands to hundreds of thousands of changes) can increase the efforts in tasks, such as change and feature propagation, quality assurance, or domain analysis. The large model differences lead to increased efforts for these tasks. Other than common variability management approaches, we were therefore interested in the feasibility of a treatment of the large model differences, that is, a treatment of the symptoms of unintentional divergence.

In this section, we describe an approach to reduce the complexity of model differences that is based on *semantic lifting* [27] and edit operation mining [54]. Semantic lifting is not a product line engineering approach but a means for handling large model differences.

4.1 Model Transformations and Edit Operations

The approach presented here is based on edit operations [27]. An *edit operation* represents a modeling operation that a user of a modeling tool can perform on a model. We consider edit operations as a special kind of model transformation [9, 16, 33]. Model transformations are a central and ubiquitous element in model-driven engineering [33, 50]. The concept of edit operations, which are endogenous (i.e., source and target meta-model are equal), in-place (i.e., the output of the model transformation overrides the input model) model transformations, can be formalized by graph transformations on typed graphs. We refer the readers to the work of Bierman et al. [9] and Ehrig et al. [16] for details. Note that, since one model revision is the result of the application of edit operations to the previous model revision, every model difference between two model revisions can be described as a (partially) ordered set of edit operations. In this context, we also speak of *change patterns* when we refer to edit operations.

4.2 Description of the Approach

The high-level idea of our approach is to simplify the presentation of model differences and to provide further insights for the user. This can speed up tasks such as change propagation. We approach the problem of large model differences by reducing the "perceived" size of the model difference. This is realized by grouping fine-grained differences to higher-level (semantic) changes. These changes can also be classified or tagged (e.g., a change can be a violation) to support the architect or developer during analysis and quality assurance related tasks. Furthermore, higher-level changes can help in the identification of common functionalities.

Technically, we achieve this by a semantic lifting of the model differences. As shown by Kehrer et al. [27], a set of defined edit operations (or high-level changes, in our case) can be recognized in a model difference, and each model difference can be expressed in terms of these high-level changes. In this semantic lifting approach, high-level changes are defined by *edit rules*, which are an executable form of edit operations. Edit rules are then lifted to recognition rules, which can be applied to a model difference to group the low-level changes in the model difference into *semantic change sets*. Furthermore, if low-level changes are contained in more than one change set, a heuristic is applied to select only disjoint change sets. Larger change sets are preferred over smaller ones. The result of lifting is a difference containing all original changes grouped into disjoint semantic change sets. For example, a change pattern that we observe quite often in the models from our case study in Section 3 is a connector and two ports added between two existing parts in an SysML internal block diagram (see Figure 2). This high-level change comprises 17 low-level changes. For many tasks related to the model differences, it will be sufficient to see this high-level information and only drill down if necessary.

In a subsequent step, filters can be applied to the lifted differences, which is especially helpful for quality assurance. Some changes will be less interesting, while others will be more interesting. For example, a renaming operation might be less interesting than adding interfaces between components. Filtering these kinds of changes improves the clarity of the model difference, especially since these changes will typically be scattered across the model difference. Furthermore, illegal changes can be highlighted in the model differences by tagging certain edit operations.

A difficulty of this approach is that the high-level changes have to be known and defined as input to the semantic lifting step. This requires a deep knowledge of the modeling language's meta-model

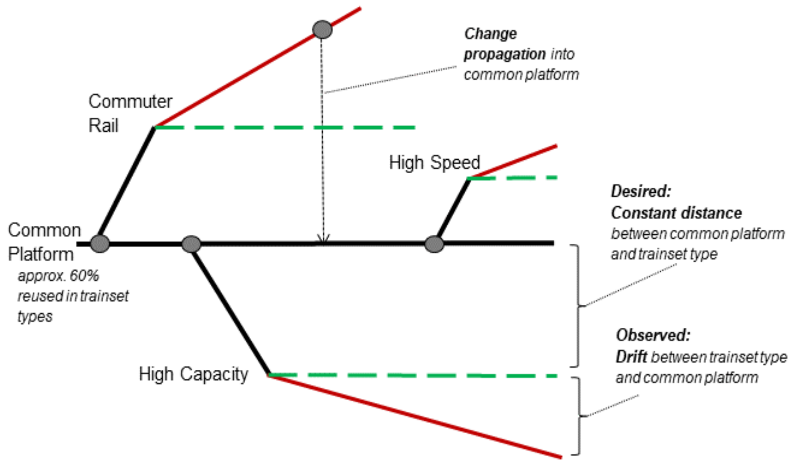
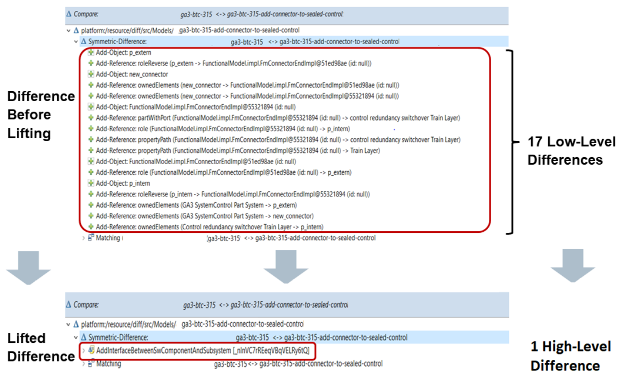

Change propagation into common platform
Commuter Rail
High Speed
Desired: Constant distance between common platform and trainset type
Common Platform
approx. 60% reused in trainset types
Observed: Drift between trainset type and common platform
High Capacity

Figure 1: Schematic representation of model drift between trainset platforms and common platform in the railway product line.



```
Δ Compare:        ga3-btc-315 <-> ga3-btc-315-add-connector-to-sealed-control
⌄ Δ platform:/resource/diff/src/Models/ ga3-btc-315-add-connector-to-sealed-control
  ⌄ Δ Symmetric-Difference:        ga3-btc-315 <-> ga3-btc-315-add-connector-to-sealed-control
      Add-Object: p_extern
      Add-Reference: roleReverse (p_extern -> FunctionalModel.impl.FmConnectorEndImpl@51ed98ae (id: null))
      Add-Object: new_connector
      Add-Reference: ownedElements (new_connector -> FunctionalModel.impl.FmConnectorEndImpl@51ed98ae (id: null))
      Add-Reference: ownedElements (new_connector -> FunctionalModel.impl.FmConnectorEndImpl@55321894 (id: null))
      Add-Object: FunctionalModel.impl.FmConnectorEndImpl@55321894 (id: null)
      Add-Reference: partWithPort (FunctionalModel.impl.FmConnectorEndImpl@55321894 (id: null) -> control redundancy switchover Train Layer)
      Add-Reference: role (FunctionalModel.impl.FmConnectorEndImpl@55321894 (id: null) -> p_intern)
      Add-Reference: propertyPath (FunctionalModel.impl.FmConnectorEndImpl@55321894 (id: null) -> control redundancy switchover Train Layer)
      Add-Reference: propertyPath (FunctionalModel.impl.FmConnectorEndImpl@55321894 (id: null) -> Train Layer)
      Add-Object: FunctionalModel.impl.FmConnectorEndImpl@51ed98ae (id: null)
      Add-Reference: role (FunctionalModel.impl.FmConnectorEndImpl@51ed98ae (id: null) -> p_extern)
      Add-Object: p_intern
      Add-Reference: roleReverse (p_intern -> FunctionalModel.impl.FmConnectorEndImpl@55321894 (id: null))
      Add-Reference: ownedElements (GA3 SystemControl Part System -> p_extern)
      Add-Reference: ownedElements (GA3 SystemControl Part System -> new_connector)
      Add-Reference: ownedElements (Control redundancy switchover Train Layer -> p_intern)
    › Matching        ga3-btc-315 <-> ga3-btc-315-add-connector-to-sealed-control

Δ Compare:        ga3-btc-315 <-> ga3-btc-315-add-connector-to-sealed-control
⌄ Δ platform:/resource/diff/src/Models/ ga3-btc-315-add-connector-to-sealed-control
  ⌄ Δ Symmetric-Difference:        ga3-btc-315 <-> ga3-btc-315-add-connector-to-sealed-control
    › AddInterfaceBetweenSwComponentAndSubsystem [_nlnVC7rREeqVBqVELRy6tQ]
    › Matching        ga3-btc-315 <-> ga3-btc-315-add-connector-to-sealed-control
```

Figure 2: Results of the semantic lifting of a connector that is added between a component and a subsystem.

(i.e., SysML, in our case) and the underlying paradigm of the transformation language. Some of the high-level changes might even be project-specific. Moreover, some of the changes are a form of tacit knowledge [42], and it will be hard for domain experts to externalize this knowledge. Leaving the definition of high-level changes to the engineers of the project causes a large overhead, which we want to avoid. In our previous work, we therefore developed an recommender system, Ockham, which is able to mine high-level change patterns from model repositories [54]. The tool is based on the so-called *minimum description length principle* [19], which states that the best *model* (in our case, a set of change patterns) to explain a *dataset* (in our case, model differences from the model repository) is the one that provides the best compression of the data. Ockham considers the problem of automatically identifying semantic change sets from a graph mining perspective. The mining of high-level changes is done in 5 steps. In Step 1, we compute model difference for successive model revisions. In Step 2, we build so called simple change graphs for these model differences. *Simple change graphs* are labeled directed graphs that break the (typically large) model differences into a set of smaller connected components.

In Step 3, *frequent subgraph mining* [13] is applied to this set of connected components for a large number of model differences. Step 4 uses a compression metric to filter for the "relevant" subgraphs and to rank the subgraphs for the output of the recommender system. In the last step, Step 5, executable edit operations for the semantic lifting approach [27] are computed from the labeled graphs.

In our previous work, we have shown our automatically discovered change patterns are relevant and help in a meaningful manner. Details of this recommender system can be found elsewhere [54], and we show there that the edit operations discovered by Ockham are meaningful for the models from the case study in Section 3.

Compared to the original semantic lifting approach [27], our combined approach *automatically* recommends meaningful edit operations for lifting model differences. Moreover, edit operations can be classified for efficient filtering of large differences.

4.3 Putting the Pieces Together

The engineering team at our industry partner uses MagicDraw for their SysML models, but there is an export to EMF. Many tools, such as code generation tools, are based on EMF. The approach presented here also builds on top of an "intermediate" EMF model. For semantic lifting, we use SiLift [27], which uses SiDiff [47] for computing structural differences. For the definition and execution of the recognition rules, SiLift uses Henshin [5]. SiLift outputs a list of changes and semantic change sets (see Figure 2), which can then be filtered and sorted. For the edit operation recommendation, we use Ockham [54], which uses SiDiff to compute model differences and Gaston [36] for frequent subgraph mining. Ockham outputs the recommended edit operations as Henshin rules.

5 CASE STUDY

In this section, we report on our case study conducted with our industry partner. We analyze the symptoms of model drift, influencing factors, and possible solutions.

Table 2: Overview of the interviewees.

Role	Scope	Experience (level)
System engineer	Single model	Entry-level
System engineer	Single model	Intermediate
Tool support	Cross-cutting	Intermediate
Chief architect	Cross-cutting	Senior
Head of tool development	Cross-cutting	Senior

5.1 Research Questions

To better understand the model drift, our goal is to answer the following research questions. RQ 1 – RQ 3 study the problem side, while RQ 4 and RQ 5 study the solution side:
RQ 1: Can model drift be observed in this product line and how is it perceived by the developers?
RQ 2: What are symptoms of model drift?
RQ 3: What are causes of model drift and large model differences?
RQ 4: Can existing solutions (i.e., bottom-up approaches and platform-based approaches) to the model drift be applied?
RQ 5: Can the semantic lifting approach support the engineers in handling large model differences?

5.2 Study Setup and Conduct

Semi-structured interviews: To get a first impression of the model drift challenge in this project, we conducted semi-structured interviews with five engineers of the project (see Table 2), among them the head of the tool development team and one of the lead architects. Three of the participants were working with more than one of the products of the product line and two of the participants did contribute to a single product. The interviews lasted 45 – 60 minutes. The interview had three sections: Challenges in clone-and-own product line engineering, unintentional divergence, and variability management solutions. In the last section, we presented the approach form Section 4 and asked the participants for their feedback regarding technical, organizational, as well as economic feasibility. We asked only open-ended questions and did not have any further constraints except covering the three sections mentioned above.

Questionnaire Survey: We then used the input from the semi-structured interview in the design of a questionnaire. We conducted a web-based[3] questionnaire survey with 9 project employees of the project from Section 3. The questionnaire was distributed via a mailing list of the architects of the project. Therefore the majority of the recipients were architects and developers involved in architectural decisions. The survey had four sections; the first section asked the interviewees about their role and experience, the second about evolution and trends of the project, the third about model differences in the project, and the last one about solution ideas and their feasibility. We gave an introduction into the topic and shortly summarized variability management solutions. We also explained the semantic-lifting approach. All together, we asked 27 questions. Since a majority of the employees are German speaking, we asked all the questions in German. The questionnaire was sent to 17 candidates and 9 filled out the questionnaire. 5 of the respondents were architects, while 4 of the respondents were developers. The project experience of the respondents ranged from 1-2 years to more than 6 years. The questionnaire is available at https://forms.office.com/r/WRvUNQisuR.

Repository data: To also get quantitative insights, we got access to the SysML models in the project. The repository used is the Teamwork Server, which is the default solution for collaborative development in MagicDraw. There is one project for each of the submodels and the repository holds the entire history for each of the submodels and all trainset platforms as well as the common platform. Table 1 shows the number of submodels, trainset platforms, as well as the average size of the models. We also used the data in this repository, to exemplary validate statements made by interviewees and survey respondents.

To answer the research questions, we use the input from the semi-structured interviews, the results from the questionnaire, as well as insights from the models in the model repository.

5.3 Results

Semi-structured interviews. During the interviews, four out of five interview partners mentioned challenges related to large model differences when asking about general challenges in their project. For example, they stated that during the propagation of changes from one product variant to another variant, the model differences can be hard to understand because of changes concurrently made by other developers. One of the interview partners did not mention challenges related to the variability at all. When we asked about the tasks and roles in the project, it turned out that she was mainly involved in the evolution of a single product and not in any activities related to variability management. As major drivers for variability, differences in hardware as well as different customer requirements have been mentioned.

All interviewees ruled out a migration to a platform-based product line engineering approach, because of migration efforts, risks, and immature tooling. Furthermore, there are less than a dozen of product variants, and therefore the effort for maintaining the platform possibly would exceed the benefits. Bottom-up approaches are conceptually more realistic, but there exists no mature tooling yet and, as a prerequisite, the variability has to be known explicitly, which is not yet the case in their setting.

The most time-consuming tasks for the interviewees are variant synchronization tasks (e.g., change propagation). Merging appears to be difficult because it is hard to define what belongs to a feature and what does not. There are many interfaces between different submodels, which further complicates merging. Computing the differences themselves is already time-consuming and takes 15 to 30 minutes per model difference. Worse, many fine-grained changes, such as renaming operations, make the model differences hard to read. This is also the case for frequently performed refactorings that are introduced because the models are adapted to new tooling. When we explained semantic lifting to the interviewees, four out of five participants stated that a semantic lifting approach has to potential speed-up several of their activities in the project. For example, the analysis of model differences for review purposes could be supported and irrelevant changes can be filtered out.

[3]We used Microsoft Forms, since it is easily accessible and employees at our industry partner are familiar with this tool.

Questionnaire. All of the answers were optional, but most of the 9 participants of our industry partner answered to all of the closed-ended questions. We also received 10 answers to open-ended questions.

78% of the respondents said they observe a drift between the different products. All of them see model drift even among the 5 biggest challenges in the project. 63% of the respondents say that, without a solution, serious problems might arise; 27% of the respondents say that a solution will increase the efficiency.

As symptoms of model drift, respondents report increasingly large model differences (78%), increased effort for change propagation (67%), increasing numbers of inconsistencies (56%), increase in communication efforts (44%), increase in feature interaction bugs (22%), and increased effort for domain analysis (33%). Furthermore, for 89% of the respondents, model differences play a role in some of their tasks in the project, for 33% even a significant role. While for only 22% of the survey participants model differences between two revisions within one product variant can be challenging, 78% of the respondents experience model differences between two product variants as challenging. For these respondents, differences play a role and can be challenging during the following tasks: reviews (88%), changes and bugfixes between two variants(100%), and domain analysis (88%). One of the participants pointed out that, even in requirements engineering (requirements are also part of models), unintended divergence is a challenge.

As causes of the model drift, all of the respondents mentioned different hardware of the products and parts of the architecture not being reused, 33% even report a significant influence of not reusing parts of the architecture. 67% think that it is difficult to define and unify features across multiple products and evolution of the single products has been mentioned by 50%. According to the respondents, large model differences are also caused by the size of the models themselves (78%), missing explicit variability information (67%), missing tool support for filtering and cherry-picking of model differences (100%), and no distinction between architectural (i.e., refactorings) and functional evolution (78%).

In Figure 3, we show a violin chart of the Likert scale data for the survey participants' opinions about 150% platform approaches, bottom-up approaches, and the semantic lifting approach. The plots shows their opinions along several dimensions: whether the approaches are suitable to provide improvement regarding the model drift challenge, whether the advantages of utilizing these approaches would outweigh the cost and risk of development and migration, whether the current tooling would support the approaches, whether the approaches are organizationally and technically feasible, and whether tooling for these approaches is known already.

Repository data. In order to also quantify the model drift, we computed the model differences between the common platform and the trainset platform for a selection of submodels and trainset platforms and three time regions (we can not take time points, because we do not have revisions at the same time points for all of the submodels). Figure 3 shows a violin plot of the size of the model differences. In this plot, we grouped the differences into three groups (Spring 2022, Spring 2021, Winter 2019). Even more, if we consider a single trainset platform and a single submodel, we can observe an increase in the size of the model difference.

We also made specific observations in the model histories, specifically with the purpose of cross-validating some of the statements of the interview and survey participants. For example, we can see that there are entire modules with the same purpose that are not reused but evolve independently for each of the trainset platforms. We also see that refactorings, such as renamings or changing the structure in the package hierarchy can obfuscate model differences. For example, the renaming of a root package will lead to a change in the *fully qualified name* of all subpackages. Also we encountered submodel revisions for which computing the model differences using MagicDraw failed because of a memory limit. We will refer to specific observations like these directly in the discussion in Section 5.4.

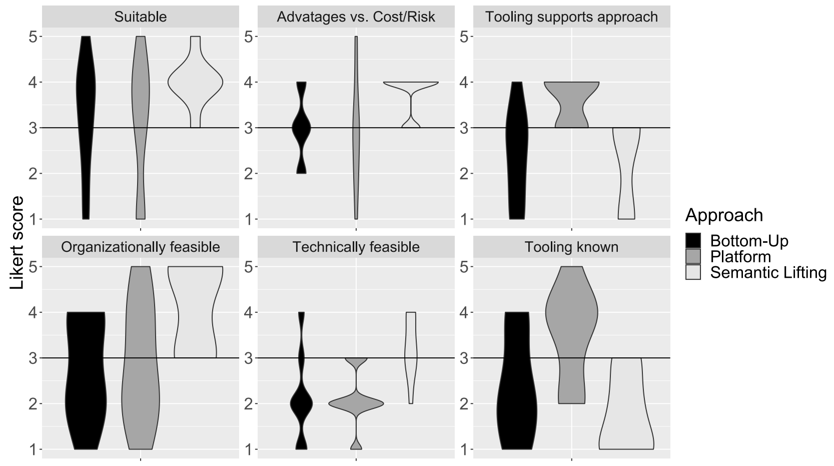
5.4 Discussion

In this subsection, we use the results from the experiments to answer the research questions from Section 5.1.

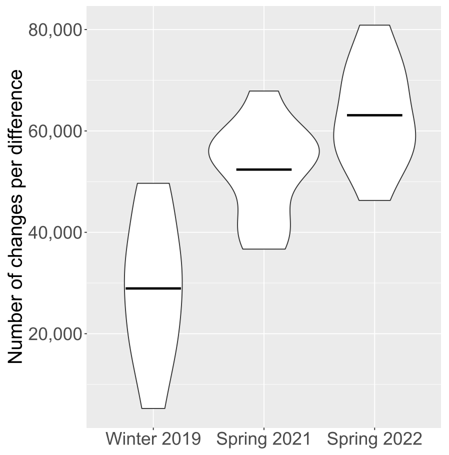
RQ 1: Model drift. Four out of five participants of the semi-structured interviews as well as 78% of the survey participants stated that they experience model drift and that model drift is a serious challenge. As can be seen from Figure 3, in terms of the size of model differences, the common platform and the trainset platform experience a drift. The time for computing the differences of the latest models in MagicDraw was 25min, on average, on a Lenovo P51 with an Intel Core™ i7-7820HQ CPU and 10GB RAM for the MagicDraw Java Application.

Summary: Model drift is perceived as serious challenge for the railways case study. Model drift is manifested in the model histories and is also perceived by a majority of the engineers from our interviews and questionnaire.

RQ 2: Symptoms of model drift. According to the interviews as well as the questionnaire, change propagation, domain analysis, and quality assurance are tasks that have become increasingly challenging because of the model drift. This is mainly because of the huge size of model differences involved in these activities. For example, for the heating, ventilation, and air conditioning model (i.e., one of the 122 submodels), MagicDraw presents 81.000 differences between one trainset platform and the common platform. This induces considerable effort for analysis tasks and variant synchronization tasks (e.g., change propagation). For instance, in our interviews, the engineers at our industry partner confirm that, on average, the analysis for quality assurance purposes takes 30 minutes per submodel. Given that often multiple submodels have to be analyzed on a regular basis — often several times per month — this leads to significant efforts and costs. Furthermore, the risk of making mistakes, for example, missing a touched *sealed component* (i.e., a component which is marked as unmodifiable), raises. If disallowed changes are discovered only late in the process – at worst during software qualification – this leads to delays in the development process which, in the end, leads to delayed delivery of the products to customers and penalties by customers. Another symptom of model drift are increased communication efforts, which were reported by 44% of the participants.



(a) Violin chart of the opinions about bottom-up approaches, platform approaches, and the semantic lifting approach. Likert scores range from 1 (=Disagree) to 5 (=Agree).



(b) Violine plot showing the number of changes per difference between common platform and trainset platforms.

Figure 3: Left: Comparison of the solution approaches. Right: Model drift in the trend of the model difference size.

Missing reuse opportunities — due to the sheer number of differences — further accelerates the drift between the different trainset platforms. This causes redundant work for the different trainset platforms. According to one of the survey participants, "rebasing of a product to a new version of the parent platform is not possible for months because the diffs are so large that tool crashes reproducibly," which then leads to some of the work been done multiple times or even new duplicates being created.

Besides the sheer number of changes, changes may be scattered across the entire model difference. As an extreme example, recently the engineering language has been changed from German to English. In total, more than 7.5% (or 98486 changes in total) of the changes are related to renaming refactorings. For some of the analysis tasks (e.g., reviews), these changes are irrelevant, but the modelling tool does not provide filters for these changes. For other tasks, such as domain engineering, it would be helpful, to classify these changes and merge them contiguously instead of being intermixed with other types of changes (e.g., functional changes).

Summary: Large model differences and increased communication efforts are the main symptoms for model drift experienced at our industry partner. Finding commonalities or reuse opportunities in these large differences has also become challenging. Mostly tasks such as change propagation, domain analysis, and quality assurance seem to be affected by the drift.

RQ 3: Causes of model drift and large model differences. We found the following drivers for model drift and large model differences.

Huge models: According to the interviewees, one reason for the enormous size of model differences is the *size of the models* themselves. The models grew steadily in size over the last seven years. The average size is 24.627 objects, where an object is an instance of a class in the meta-model. That is, attributes and references are not even included in this count. The reason is that the train software and system is modeled with a high degree of detail. Various additional information is provided, for example, requirements traceability or organizational information.

Duplicated architecture: In the model repositories, we found parts of the models that seem to have a similar purpose but are modelled individually for every trainset platform. These duplicated parts appear in every model difference and also inflate the number of changes in these differences. Different hardware used for the different trainset platforms appears to be one of the main drivers for inducing new variability and duplicated parts of the models. The interviews and survey support this finding. One of the survey respondents said that "under time pressure, often only a single solution is sought for the concrete problem in the concrete project instead of a solution suitable for the platform."

Implicit variability: Many of the problematic changes that are typically found during quality assurance are related to missing explicit constraints related to variability. For example, there are components which must not be modified in one of the trainset platforms. Because this information is often implicit, trainset platform engineers sometimes change parts of a model where actually no change should occur. Missing variability information increases the size of the model differences: Features that are only part of the platform but not present in a specific train type will be always shown in a model difference, even though they might not be relevant. Likewise, features that are only part of the specific trainset platforms but irrelevant to the platform should be excluded from the platform when merging train type changes. For tasks such as extracting commonalities, such features are of special interest. This extraction would be easier if only product-specific features are compared across the different train types, that is, explicit variability should be used for isolated feature development.

Missing tool support: Especially for domain analysis, a model difference between a trainset platform and the common platform or between two trainset platforms is computed. In the differences, there will be many changes that are not relevant for the domain analysis. One limitation of MagicDraw is that it does not support filtering model differences. Even given a concrete feature mapping, MagicDraw does not provide any filtering functionality out-of-the box. This is also partly related to the fact that there is no commonly accepted way to express variability in SysML models. The engineers at our industry partner not only need support for Boolean features but also numeric and topological features. Furthermore, MagicDraw only supports two-way and three-way merging to propagate the changes between branches. Merging aims at including as many changes as possible, while in different variants, some differences are desired [28]. *Cherry-picking* changes (i.e., only selecting changes between two consecutive revisions) would support change propagation, but MagicDraw does not support cherry-picking as of now. On of the survey respondents mentioned that "poor tool support for taking over simple changes leads to the same changes being brought in individually and thus only being the same, but not the same elements, so that technically further diffs arise." In our repository data, we can also see these duplicated blocks, which will show up in the model differences.

Classification of changes: Model differences typically show many fine-grained changes. Some of these fine-grained changes belong to higher-level changes, but they are not grouped together in the presentation of model differences. This increases the "perceived" size of the model differences. Another challenge that increases the complexity of model differences is the lack of a distinction between architectural evolution (i.e., refactoring) and functional evolution. For example, correcting a typo in a component name is presented in a similar way to the user as changing a connector. Another example are move refactorings, for example, a package is moved in the containment hierarchy, or graphical changes in diagrams. As one of the survey respondents states, "[...] semi-automatic changes (example: upper/lower case of elements) lead to HUGE diffs, so that no manual diff (and merge) is possible for a long time." These changes are typically scattered across a model, and they are ubiquitous. For some activities, such as quality assurance, it would be helpful to distinguish these changes from other, more important changes.

Summary: Large models together with a lack of support for explicit variability as well as missing tool support for filtering model differences result in large numbers of model differences that have to be analyzed. Therefore, it is increasingly difficult to reuse parts of the system across multiple trainset platforms. Parts of the models can not be reused even though they have a similar purpose. These factors cause the variants drifting apart over time.

RQ 4: Solution approaches. During our interviews and in the survey, we identified several hindrances using existing solution approaches (see Section 2.2 and Section 2.3), which we discuss in this subsection w.r.t. possible solution approaches.

Platform approaches: Our interviewees and survey participants mentioned several barriers to use 150% platform approaches. Most prominently, platform approaches can not be used because of the cost and risk of a migration. At our industry partner, there is already a significant amount of modelling data that would have to be migrated. One of the participants of the interview mentioned that "many of the developers think in terms of concrete products and not the entire product line". This also shows that a migration from pure clone-and-own to a (150% platform) software product line approach will face organizational hindrances. Furthermore, there is no variability modelling approach that supports Boolean constraints, numeric constraints, as well as also topological constraints. One of the survey participants even said that "the structural variability is so large that it can not be modelled using a 150% platform." Some mentioned that such an approach could be useful as a basis (which is also a goal of current efforts) but some of the differences will be too special and need individual solutions.

Bottom-up approaches: According to our interviews, a support for understanding the variability and commonalities between the different trainset platforms is clearly desirable. Nevertheless, according to interviews and survey, approaches proposed in the literature are not mature enough. For example, as for the 150% platform approaches, there is no approach that supports Boolean constraints, numeric constraints, and topological constraints. Additionally, variation control systems proposed for models (e.g., SuperMod [49]) have not yet been evaluated in an industrial setting. This is complicated by the fact that there is no universally accepted standard for expressing variability in modeling languages [8].

Typically, bottom-up approaches make use of traces between features and implementation artifacts. Automatically reverse engineering the feature traces in this product line is not feasible. This is not only because of non-Boolean constraints, but also because of the size of the models (and therefore scalability issues), and because these approaches discover many *merged feature candidates* that contain more than one feature. Furthermore, there are not enough products available to discover a useful set of feature candidates. In general, the "development of methodology and tooling in the domain would be of enormous complexity", according to a survey respondent. Nevertheless, the engineering team at our industry partner developed an approach to feature traceability, and the manual incorporation of explicit variability information into the models is an ongoing activity. Still, there are only a few feature traces yet which could be used.

Summary: Common variability management solutions proposed in the literature are not applicable to the setting at our industry partner. This is mainly due to immaturity of the available tooling, but also because many approaches make simplifying assumptions, not taking real-world and organizational constraints into account. A migration to another approach comes with high costs and risk, in any case.

RQ 5: Semantic lifting. While platform-based and bottom-up approaches are not (yet) applicable in this setting, the results from the interviews and the questionnaire survey suggest that the semantic lifting approach from Section 4 appears to be suitable to support the engineers in handling large model differences.

Having a *semantic change log* for model differences "has already been hypothesized as a valuable solution in a workshop", according to one of the interview participants. Four out of five interviewees stated that using higher-level change patterns in the presentation of model differences would support many tasks, such as merging features, filtering the differences during reviews, identifying illegal changes in model differences, and understanding model differences in general. The remaining participant did not perform any tasks where the current size of the model difference was considered as a problem. Two interviewees, who were responsible for propagating changes between the common platform and trainset platforms described the merge process as one of their biggest challenge: since they "[...] work with several workers in the project, it is also very time-consuming [during merge] to filter out what my own changes were. [...] Usually several points in the model are affected by a small change — external interfaces, software, product structure, physical structures — so there are changed elements in every folder, so to speak." Grouping related changes can support here as well. Two of the interviewees mentioned that change patterns could also be used to easier recognize complex refactorings in model differences. In earlier work[4], it has been shown that, in this product line, recommended edit operations for the semantic lifting achieve a compression ratio of about 4.0, on average (without filtering yet).

The current tooling does not support such an approach natively but, according to one of the interview participants, it would be possible to develop a plugin for MagicDraw. Unlike the bottom-up approaches, semantic lifting appears to be more feasible from a technical viewpoint (see Figure 3). Nevertheless, one of the survey participants is also concerned that the "[...] current lack of experience in that field, could be an obstacle in the development of a production-ready solution."

Interview and survey participants at our industry partner largely support the hypothesis that a semantic lifting approach mitigates challenges arising from the model drift. The hypothesis is further supported by the observation that the differences can indeed be compressed using semantic lifting in this product line. Nevertheless, there are concerns that the production-ready implementation of a semantic lifting approach can be challenging.

5.5 Threats to Validity

The purpose of this work is to report on a concrete large-scale product line in an industrial setting. We do not claim the generalizability to other large-scale product lines here, although we strongly believe that similar challenges exist in many industrial settings. This is also suggested by the work of Dubinsky et al. [14] and Rubin et al. [44].

A threat to internal validity is caused by our setup of the interviews and the questionnaire. We had no control group, because we only had a small group of participants. Nevertheless, in particular regarding the existence of the drift, we used three different methods (interviews, survey, and repository data) to validate the presences of an unintentional divergence. Furthermore, we did not evaluate the semantic lifting approach in a long-term application at our industry partner. For future work, we need to put the approach to the acid test by letting the engineers work with the tooling.

A threat to external validity is that we only used a small sample of engineers from the project for our interviews and in the survey. Nevertheless, the engineers participating in the interviews and the survey were involved into architectural decisions or had an otherwise leading role. We also used only a subset of all submodels in the project for our repository analysis. Anyhow, because the results of interviews, survey, and the repository analysis are consistent, we are confident that our results hold for the entire project. Also, many results were evident from a small sample set, for example, we could find the drift for all single products in the sample.

6 CONCLUSION AND OUTLOOK

In this paper, we report on a study on model drift in an industrial large-scale model-driven clone-and-own software product line. We conducted semi-structured interviews, a questionnaire survey, and we analyzed the model repository to understand model drift in this product line. We found substantial model drift that impairs several tasks, such as change propagation, domain analysis, and quality assurance. The main reason is that these tasks are based on large model differences that have to be analyzed manually by engineers. We found that solutions described in the literature are often not applicable in a real-world setting, for example, filtering model differences for certain features requires a rigorous modelling of variability in the models and existing tools miss some critical functionality, for example, handling structural variability. Instead of addressing variability management as one of the root causes for large model differences, we also studied the alternative route of addressing large model differences directly. Based on our observations, we proposed a concrete solution to mitigate the problem by summarizing the model differences in terms of higher-level change patterns that are automatically derived from model repositories based on the idea that meaningful patterns are the ones that compresses model differences. In the context of the project at our industry partner, this idea appears to hold true. Our mixed methods study provides evidence that semantic lifting can support engineers in tackling large model differences. The approach is orthogonal to variability management solutions. For example, even using variability management solutions, model differences can still be large (e.g., because of the size of the models or missing tool support for filtering model differences properly).

Other existing solutions to support software product-line engineering related tasks require explicit variability in the models. Establishing a bottom-up, managed cloning product line engineering is a long-term target at our industry partner. The engineering team at our industry partner is currently working on making variability explicit by creating explicit mappings from features to model elements. As a long-term solution, it is conceivable to use this variability information, for instance, to propagate changes between trainset platforms or for filtering of models and model differences.

While in the long-run it is imperative to establish a working variability management approach, in the short-term, treating symptoms of unintentional divergence can support engineers and avoids costs and risk of more costly solutions.

ACKNOWLEDGEMENTS

We thank all the participants of the survey and the interviews as well as all reviewers for their valuable feedback. Apel's work has been supported by DFG grant AP 206/14-1 and DFG Grant 389792660 as part of TRR 248 – CPEC.

[4]An extension of [54] is currently under review and available as preprint at https://www.researchsquare.com/article/rs-1558716/v1

REFERENCES

[1] Kerstin Altmanninger, Kerstin Kappel, Angelika Kusel, Werner Retschitzegger, Martina Seidl, Wieland Schwinger, and Manuel Wimmer. 2008. AMOR – Towards Adaptable Model Versioning. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS) - International Workshop on Model CoEvolution and Consistency Management*. 55–60. https://doi.org/10.1.1.159.3404

[2] Sven Apel, Don S. Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines - Concepts and Implementation*. Springer. https://doi.org/10.1007/978-3-642-37521-7

[3] Sven Apel, Florian Janda, Salvador Trujillo, and Christian Kästner. 2009. Model superimposition in software product lines. In *International Conference on Theory and Practice of Model Transformations*. Springer, 4–19. https://doi.org/10.1007/978-3-642-02408-5_2

[4] Sven Apel, Christian Kastner, and Christian Lengauer. 2009. Featurehouse: Language-independent, automated software composition. In *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 221–231. https://doi.org/10.1109/ICSE.2009.5070523

[5] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. 2010. Henshin: Advanced concepts and tools for in-place EMF model transformations. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS)*. Springer, 121–135. https://doi.org/10.1007/978-3-642-16145-2_9

[6] Thorsten Arendt and Gabriele Taentzer. 2013. A tool environment for quality assurance based on the Eclipse Modeling Framework. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. IEEE/ACM, 141–184. https://doi.org/10.1007/s10515-012-0114-7

[7] Don Batory, Jacob Neal Sarvela, and Axel Rauschmayer. 2004. Scaling stepwise refinement. *Transactions on Software Engineering* 30, 6 (2004), 355–371. https://doi.org/10.1109/TSE.2004.23

[8] Thorsten Berger, Jan-Philipp Steghöfer, Tewfik Ziadi, Jacques Robin, and Jabier Martinez. 2020. The state of adoption and the challenges of systematic variability management in industry. *Empirical Software Engineering* (2020). https://doi.org/10.1007/s10664-019-09787-6

[9] Enrico Biermann, Claudia Ermel, and Gabriele Taentzer. 2012. Formal foundation of consistent EMF model transformations by algebraic graph transformation. *Software and Systems Modeling* 11, 2 (2012), 227–250. https://doi.org/10.1007/s10270-011-0199-7

[10] CENELEC/EN. 2001. *50126: Railway applications-The specification and demonstration of Reliability*. Technical Report. CENELEC/EN.

[11] CENELEC/EN. 2003. *50129: Railway application–Communications, signaling and processing systems–Safety related electronic systems for signaling*. Technical Report.

[12] CENELEC/EN. 2012. *50128: Railway applications-Communication, signalling and processing systems-Software for railway control and protection systems*. Technical Report. CENELEC/EN.

[13] Diane J Cook and Lawrence B Holder. 2006. *Mining graph data*. John Wiley & Sons.

[14] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. 2013. An exploratory study of cloning in industrial software product lines. In *Proceedings of the European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE, 25–34. https://doi.org/10.1109/CSMR.2013.13

[15] Jorge Echeverría, Francisca Pérez, José Ignacio Panach, and Carlos Cetina. 2021. An empirical study of performance using Clone & Own and Software Product Lines in an industrial context. *Information and Software Technology* 130 (2021). https://doi.org/10.1016/j.infsof.2020.106444

[16] Hartmut Ehrig, Ulrike Prange, and Gabriele Taentzer. 2004. Fundamental theory for typed attributed graph transformation. In *Lecture Notes in Computer Science*, Vol. 3256. 161–177. https://doi.org/10.1.1.1067.9010

[17] Tzilla Elrad, Omar Aldawud, and Atef Bader. 2002. Aspect-oriented modeling: Bridging the gap between implementation and design. In *International Conference on Generative Programming and Component Engineering*. Springer, 189–201. https://doi.org/10.1007/3-540-45821-2_12

[18] Stefan Fischer, Lukas Linsbauer, Roberto E. Lopez-Herrejon, and Alexander Egyed. 2016. A vision for enhancing clone-and-own with systematic reuse for developing software variants. *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI)* P252 (2016), 95–96. https://doi.org/20.500.12116/741

[19] Peter D. Grünwald and Abhijit Grunwald. 2007. *The minimum description length principle*. MIT Press. https://doi.org/10.5555/1213810

[20] IEC. 2010. *61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems*. Technical Report. IEC.

[21] No Magic Inc. 2021. *MagicDraw website*. Retrieved September 20, 2021 from https://www.magicdraw.com/

[22] Praveen Jayaraman, Jon Whittle, Ahmed M Elkhodary, and Hassan Gomaa. 2007. Model composition in product lines and feature interaction detection using critical pair analysis. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS)*. Springer, 151–165. https://doi.org/10.1007/978-3-540-75209-7_11

[23] Hans Peter Jepsen, Jan Gaardsted Dall, and Danilo Beuche. 2007. Minimally Invasive Migration to Software Product Lines. In *Proceedings of the Software Product Line Conference (SPLC)*. 203–211. https://doi.org/10.1109/SPLINE.2007.30

[24] Cory J Kapser and Michael W Godfrey. 2008. "Cloning considered harmful" considered harmful: patterns of cloning in software. *Empirical Software Engineering* 13, 6 (2008), 645–692. https://doi.org/10.1007/s10664-008-9076-6

[25] Christian Kästner, Salvador Trujillo, and Sven Apel. 2008. Visualizing Software Product Line Variabilities in Source Code. In *Proceedings of the Software Product Line Conference (SPLC)*. 303–312. https://doi.org/10.1.1.143.2730

[26] Timo Kehrer. 2015. *Calculation and Propagation of Model Changes based on User-Level Edit Operations: A Foundation for Version and Variant Management in Model-Driven Engineering*. Ph.D. Dissertation. University of Siegen.

[27] Timo Kehrer, Udo Kelter, and Gabriele Taentzer. 2011. A rule-based approach to the semantic lifting of model differences in the context of model versioning. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. ACM/IEEE, 163–172. https://doi.org/10.1109/ASE.2011.6100050

[28] Timo Kehrer, Thomas Thüm, Alexander Schultheiß, and Paul Maximilian Bittner. 2021. Bridging the Gap Between Clone-and-Own and Software Product Lines. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE/ACM, 21–25. https://doi.org/10.1109/ICSE-NIER52604.2021.00013

[29] Amal Khalil, Juergen Dingel, and Canada Kl. 2013. Supporting the Evolution of UML Models in Model Driven Software Development: A Survey Technical Report 2013-602 School of Computing , Queen' s University. February (2013).

[30] Jacob Krüger and Thorsten Berger. 2020. An empirical analysis of the costs of clone- and platform-oriented software reuse. In *Proceedings of the Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 432–444. https://doi.org/10.1145/3368089.3409684

[31] Lukas Linsbauer, Thorsten Berger, and Paul Grünbacher. 2017. A classification of variation control systems. *ACM SIGPLAN Notices* 52, 12 (2017), 49–62. https://doi.org/10.1145/3136040.3136054

[32] Jabier Martinez, Wesley K. G. Assunção, and Tewfik Ziadi. 2017. ESPLA: A Catalog of Extractive SPL Adoption Case Studies. In *Proceedings of the Software Product Line Conference (SPLC)*. ACM, 38–41. https://doi.org/10.1145/3109729.3109748

[33] Tom Mens and Pieter Van Gorp. 2006. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science* 152, 1-2 (2006), 125–142. https://doi.org/10.1016/j.entcs.2005.10.021

[34] Tom Mens, Michel Wermelinger, Stéphane Ducasse, Serge Demeyer, Robert Hirschfeld, and Mehdi Jazayeri. 2005. Challenges in software evolution. In *8th International Workshop on Principles of Software Evolution (IWPSE'05)*. IEEE, 13–22. https://doi.org/10.1109/IWPSE.2005.7

[35] Andreas Metzger and Klaus Pohl. 2014. Software product line engineering and variability management: achievements and challenges. In *Proceedings of the Conference on Future of Software Engineering (FOSE)*. ACM, 70–84. https://doi.org/10.1145/2593882.2593888

[36] Siegfried Nijssen and Joost N. Kok. 2005. The Gaston tool for frequent subgraph mining. *Electronic Notes in Theoretical Computer Science* 127, 1 (2005), 77–87. https://doi.org/10.1016/j.entcs.2004.12.039

[37] Richard F Paige, Nicholas Matragkas, and Louis M Rose. 2016. Evolving models in Model-Driven Engineering: State-of-the-art and future challenges. *Journal of Systems and Software* 111 (2016), 272–280. https://doi.org/10.1016/j.jss.2015.08.047

[38] Tristan Pfofe, Thomas Thüm, Sandro Schulze, and Ina Schaefer. 2016. Synchronizing software variants with VariantSync. In *Proceedings of the Software Product Line Conference (SPLC)*. ACM, 329–332. https://doi.org/10.1145/2934466.2962726

[39] Christopher Pietsch, Timo Kehrer, Udo Kelter, Dennis Reuling, and Manuel Ohrndorf. 2015. SiPL–A Delta-Based Modeling Framework for Software Product Line Engineering. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. IEEE, IEEE, 852–857. https://doi.org/10.1109/ASE.2015.106

[40] Christopher Pietsch, Udo Kelter, Timo Kehrer, and Christoph Seidl. 2019. Formal foundations for analyzing and refactoring delta-oriented model-based software product lines. In *Proceedings of the Software Product Line Conference (SPLC)*. 207–217. https://doi.org/10.1145/3336294.3336299

[41] Klaus Pohl, Günter Böckle, and Frank Linden. 2005. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer. https://doi.org/10.1007/3-540-28901-1

[42] Michael Polanyi. 1958. *Personal Knowledge: Towards a Post Critical Philosophy*. University of Chicago Press. 428 pages.

[43] Louis Mathew Rose. 2011. *Structures and Processes for Managing Model-Metamodel Co-evolution*. Ph.D. Dissertation. University of York.

[44] Julia Rubin, Krzysztof Czarnecki, and Marsha Chechik. 2013. Managing cloned variants: A framework and experience. In *Proceedings of the Software Product Line Conference (SPLC)*. ACM, 101–110. https://doi.org/10.1145/2491627.2491644

[45] Julia Rubin, Andrei Kirshin, Goetz Botterweck, and Marsha Chechik. 2012. Managing forked product variants. In *Proceedings of the Software Product Line Conference (SPLC)*. ACM, 156–160. https://doi.org/10.1145/2362536.2362558

[46] Ina Schaefer. 2010. Variability Modelling for Model-Driven Development of Software Product Lines. *VaMoS* 10 (2010), 85–92.

[47] Maik Schmidt and Tilman Gloetzner. 2008. Constructing difference tools for models using the SiDiff framework. In *Proceedings of the International Conference on Software Engineering (ICSE): Companion Proceedings*. ACM/IEEE, 947–948. https://doi.org/10.1145/1370175.1370201

[48] Thomas Schmorleiz and Ralf Lämmel. 2014. Similarity management via history annotation. *SATToSE* (2014), 45.

[49] Felix Schwägerl. 2018. *Version Control and Product Lines in Model-Driven Software Engineering*. Ph.D. Dissertation.

[50] Shane Sendall and Wojtek Kozaczynski. 2003. Model transformation: The heart and soul of model-driven software development. *IEEE Softw.* 20, 5 (2003), 42–45. https://doi.org/10.1109/MS.2003.1231150

[51] Stefan Stănciulescu, Thorsten Berger, Eric Walkingshaw, and Andrzej Wąsowski. 2016. Concepts, operations, and feasibility of a projection-based variation control system. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 323–333. https://doi.org/10.1109/ICSME.2016.88

[52] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. 2008. *EMF: eclipse modeling framework*. Pearson Education.

[53] Ragnhild Van Der Straeten, Tom Mens, and Stefan Van Baelen. 2009. Challenges in model-driven software engineering. In *Lecture Notes in Computer Science*, Vol. 5421. 35–47. https://doi.org/10.1007/978-3-642-01648-6_4

[54] Christof Tinnes, Timo Kehrer, Joblin. Mitchell, Uwe Hohenstein, Andreas Biesdorf, and Sven Apel. 2021. Learning domain-specific edit operations from model repositories with frequent subgraph mining. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. ACM/IEEE. https://doi.org/10.1109/ASE51524.2021.9678698

[55] Birgit Vogel-Heuser, Christoph Legat, Jens Folmer, and Stefan Feldmann. 2014. *Researching evolution in industrial plant automation: Scenarios and documentation of the pick and place unit*. Technical Report. Institute of Automation and Information Systems, Technische Universität München.

[56] Nathan Weston, Ruzanna Chitchyan, and Awais Rashid. 2009. A framework for constructing semantically composable feature models from natural language requirements. In *Proceedings of the Software Product Line Conference (SPLC)*. 211–220. https://doi.org/10.5555/1753235.1753265

[57] Kentaro Yoshimura, Dharmalingam Ganesan, and Dirk Muthig. 2006. Assessing merge potential of existing engine control systems into a product line. In *International Workshop on Software Engineering for Automotive Systems*. 61–67. https://doi.org/10.1145/1138474.1138485

[58] Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. 2019. What the fork: A study of inefficient and efficient forking practices in social coding. In *Proceedings of the Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) (ESEC/FSE 2019)*. ACM, New York, NY, USA, 350–361. https://doi.org/10.1145/3338906.3338918